MAC OS RUNTIME
FOR JAVA

Using JBindery

JBindery 2.0.1
For MRJ 2.0

6/19/98
Technical Publications

Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Using JBindery

Contents

JBindery is an application that you use to package or execute Java™ applications on the Mac OS platform. You can use JBindery to do any of the following:

- package Java applications so that they can be launched like any Mac OS application
- execute previously compiled Java applications or applets
- save settings (for example, parameter lists or security levels) for frequently run Java applications

For a listing of changes from the previous version of JBindery, see "Changes from Version 2.0" (page 6).

How to Use This Document

To use JBindery, you first should read the section "JBindery Features" (page 7). Then you can use the "recipes" that follow in the section "Examples" (page 21) to package or execute your Java application.

This document assumes that you have general knowledge of the Java language and terminology ("virtual machine," "properties," and so on). For further information about the Java language or low-level details of the Java virtual machine, you should consult JavaSoft documentation, which you can access through the JavaSoft home page:

http://java.sun.com/

You do not need any Mac OS programming experience to use JBindery. However, in cases where you might want to add Mac OS–specific features to your standalone Java application (for example, a custom desktop icon), you may need to use some Mac OS development tools (such as the ResEdit resource editor). To create polished applications with the proper Mac OS look and feel, you must follow the specifications provided in the book *Macintosh Human Interface Guidelines* available at the following Web site:

http://www.apple.com/developer/

For more information about Apple's use of Java technology, see the following Web page:

http://developer.apple.com/java/

Changes from Version 2.0

Although the basic functionality of JBindery is the same, this version includes some organizational changes and a few new features:

- The functionality of the Virtual File System panel is now combined with the Classpath panel.
- JBindery now has a preferences file which can specify default settings to use at startup.
- JBindery now prompts you to save settings changes before quitting.

The new version of JBindery cannot be used with versions of MRJ prior to 2.0. However, applications built with older versions of JBindery will still run under MRJ 2.0.

Note
JBindery 2.0 (released with MRJ 2.0) did not contain any feature enhancements over the original version of JBindery released with MRJ 1.5 ◆

System Requirements

JBindery uses the Mac OS Runtime for Java (MRJ) environment to execute Java applications. To use JBindery, your system must have MRJ 2.0 or later installed. You can download the most recent version of MRJ from the Apple Java Web page:

http://developer.apple.com/java/

To use MRJ 2.0, your computer must have a 68040 or PowerPC microprocessor with at least 24 MB of RAM, 16 MB of which must be physical (that is, not virtual memory), and 13 MB of free disk space. System 8.0 or later is suggested, although you can install MRJ 2.0 on System 7.6.1 by specifying a custom install. Computers running with a 68040 microprocessor must have 32-bit addressing turned on (as specified in the Memory control panel).

How JBindery Works

Java programs must run in a virtual machine that is created on the host platform. The virtual machine uses services provided by the host platform to map graphical information provided by the Abstract Window Toolkit (AWT) to the user-visible screen and to pass any user input to the Java program. JBindery provides these services to your Java program. For example, if you use JBindery to execute a Java application, JBindery instantiates a Java virtual machine and then calls the Java application's `main` method. After the Java application quits, JBindery performs cleanup (removing the virtual machine and so on) and quits.

If you create a standalone Java application, JBindery packages a small wrapper Mac OS application in the Java application file. Creation and removal of the virtual machine is then handled transparently when you execute the packaged Java application.

Note
JBindery relies on the JManager function library (as provided with MRJ 2.0 and later) to interact with Java programs. For detailed information about these Mac OS functions, see the document *Programming with JManager*. ◆

JBindery Features

Launching JBindery brings up a number of panels that contain information about the Java program you want to execute. To switch among the various panels, select any of the panel icons, which are displayed along the left side of any panel.

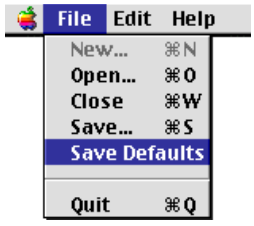
Many settings on the JBindery panels correspond to options used on operating systems that have a command line interface. For example, the options in the UNIX command line

```
java MyClass -classpath myDirectory:MyClasses -Dbanana=yellow -verify
```

can be set in the Classpath, Properties, and Security panels in JBindery.

If you want to change the default settings that appear when you launch JBindery, you can do so by modifying the settings and selecting the Save Defaults menu item as shown in Figure 1-1.

Figure 1-1 The Save Defaults menu item

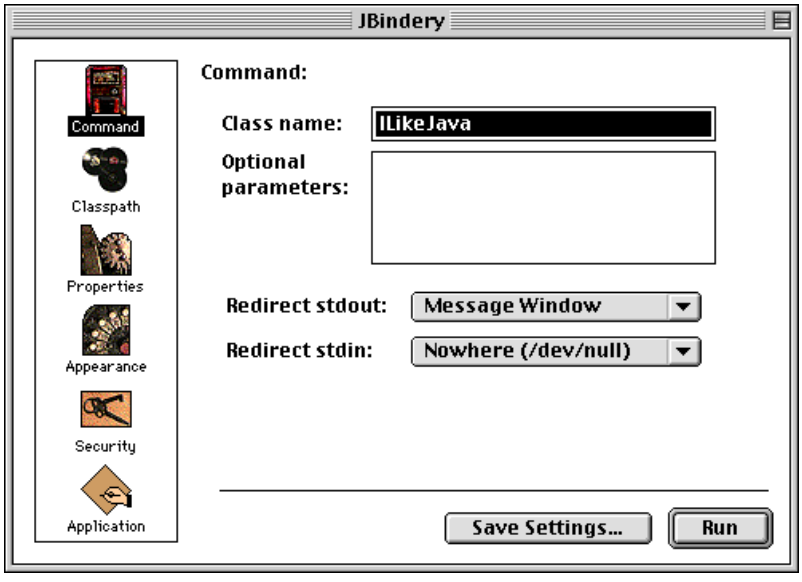


The new default settings are stored in the file JBindery Prefs in the Preferences folder. You can restore the original default settings by throwing away the preferences file.

The Command Panel

Figure 1-2 shows the Command panel of JBindery. This is the default panel that appears when you launch JBindery.

Figure 1-2 The Command panel



The Command panel determines the Java class to execute, the parameters to pass to the class's main method, and the locations to direct console input or output.

- The Class Name field should contain the name of the class you want to execute. This name should have the form `package.package.classname` without a `.class` extension. If desired, you can specify a slash (`/`) instead of the period (`.`) as a delimiter.

  The class you specify must contain a method with one of the following declarations:

  ```
  public static void main (String args[])
  public static void main()
  ```

  If you launch JBindery by dragging a class file onto the JBindery icon, JBindery assumes the class name is the name of the file without the extension, and places this name in the Class Name field. For example, if the file is `biology.class`, the class name is assumed to be `biology`. If the name of the file does not correspond to the class containing the main method, you should enter the class name manually.

- The Optional Parameters field lets you specify text parameters to pass to the main method. JBindery makes assumptions based on the parameters as follows:
  - If no parameters are specified, JBindery attempts to execute the `main()` method with no arguments. If no `main()` method is found in the class, JBindery creates a zero-length string array and calls the `main (String args[])` method.
  - If you specify parameters, JBindery attempts to execute the `main (String args[])` method. If no `main (String args[])` method exists in the class, then JBindery calls the `main()` method with no parameters.
  - If JBindery cannot find any `main` method, it throws an exception.

  JBindery considers spaces, tabs, and carriage returns as parameter delimiters. If you want to include such characters as part of a parameter, you must enclose the string in quotation marks (`" "`).

  If you want to include quotation marks in your parameter, you must precede each instance with a backslash (`\`). For example, you would specify the string `He said, "Hi"` as `"He said, \"Hi\""`.

  To specify Unicode strings, you should precede the Unicode value with `\x`.
- The Redirect Stdout pop-up menu lets you redirect any console output (that is, any output that the Java application writes to `System.out` or `System.err`). Selecting the pop-up menu displays the following options:
  - Message Window: Console output appears in a plain text window.
  - Nowhere: Console output is ignored.
  - To File...: Console output is sent to a new file, overwriting an existing file if necessary. Selecting this option brings up a save dialog box that allows you to specify the name and location of the output file.
  - Append File...: Console output is appended to an existing file (or a new one if it does not currently exist). Selecting this option brings up a dialog box that allows you to specify the name and location of the output file.
- The Redirect Stdin pop-up menu specifies the source of any console input (that is, any input that comes from `System.In`). Selecting the pop-up menu displays the following options:
  - Nowhere: No input is taken.
  - Message Window: The user is prompted to enter text in a plain text window. This is the same window that displays console output.

- From File: The specified file is treated as the input source. Selecting this option brings up a dialog box that allows you to specify the name and location of the input file.

Along the bottom of the panel there are two buttons you use to control JBindery actions. These buttons appear on every panel.

- The Save Settings button brings up a save dialog box. You use this button to save your application settings as a settings file or as part of a packaged application.
- The Run button executes your Java application with the specified settings.

The Classpath Panel

Figure 1-3 shows the Classpath panel of JBindery, which displays the class paths used to search for Java classes.

By default, JBindery searches the `MRJ Libraries` folder when looking for classes. However, you can use the Classpath panel to specify additional class paths in a manner similar to setting the `CLASSPATH` Java environment variable. Any class paths you specify do not replace the default, but are appended to the class path list. These path entries are searched in the order they appear in the list. The `MRJ Libraries` folder is always searched last.

Figure 1-3 The Classpath panel

You can add any of the following entries to the class path list:

- An alias to a folder. An alias is a reference that lets you access a file or a folder stored in another location. See "Adding an Alias" (page 13) for more information.
- An alias to a .zip or .jar package
- An absolute path to a file, folder, or package
- A path to a file, folder, or package relative to the application.
- A virtual file system. A virtual file system is useful only if you want to package all the files that make up your Java application in one package. See "Adding a Virtual File System" (page 14) for more information.

Note that if your class files have names longer than the 31 characters allowed by the Mac OS, you can bypass this limitation by storing them in a .zip or .jar archive.

You can change the search position of any item in the class path list by selecting it and then using the up and down arrows. You can remove a selected item by

using the Delete button. Clicking the Edit… button allows you to edit the current selection.

Adding an Alias

An alias is a small file that represents another file stored in a different location. Typically, you use them to simplify organizing files or to allow easy access to large or dynamic files that reside on a file server. In most cases, you can handle an alias just as you would the original file. For example, you could open the original file by double-clicking on its alias.

Aliases provide a quick and easy way to set up class paths if you plan to use JBindery to simply execute a Java application (for example, during development). However, aliases are generally not portable. Since aliases often store information about the local file system (the name of the hard drive, and so on), the Finder may not be able to resolve the alias if it is moved to another computer. For this reason, you should not add class paths as aliases if you plan to run your Java application on multiple computers.

You can add an alias to your class path using any of the following methods:

- Using the Add Folder button. Doing so brings up a dialog box that lets you specify a folder to add to the class path. The folder you choose must contain class files or be the root of a directory hierarchy that mirrors a class hierarchy.
- Using the Add .zip File button. Doing so brings up a dialog box that lets you specify an uncompressed .zip or .jar file that contains class files.
- Drag a class file, folder, or .zip file onto the class path list from the Finder. JBindery automatically adds the parent folder of the class file to the list as an alias. Similarly, folders or .zip files are added to the list as aliases. You can also add aliases by dragging a class file or package onto the JBindery icon.

Adding a Class Path Manually

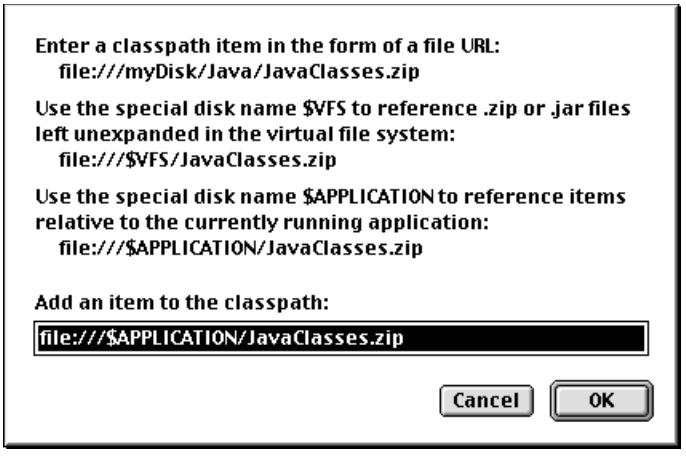
The Add Manually button brings up a dialog box as shown in Figure 1-4. You must specify the location of a folder or an uncompressed .zip or .jar file as a file-based URL . You can also use this dialog box to specify unexpanded .zip or .jar files contained in a virtual file system, which is described in “Adding a Virtual File System” (page 14).

The disk name `/$APPLICATION/` is an Apple-specific designation that you can use in your file-based URL to indicate the folder containing the running application. For example, if you wanted to include a folder `NewClasses`

contained in application's folder, you would designate the URL as `file:///$APPLICATION/NewClasses`. The `/$APPLICATION/` designation corresponds to the location of JBindery or the packaged Java application, not the location of the Java classes.

Adding classes manually allows for greater portability across computers than using aliases. The class path `file:///$APPLICATION/NewClasses` will always work as long as the `NewClasses` folder is in the same folder as the application.

Figure 1-4 The Add Manually dialog box



Adding a Virtual File System

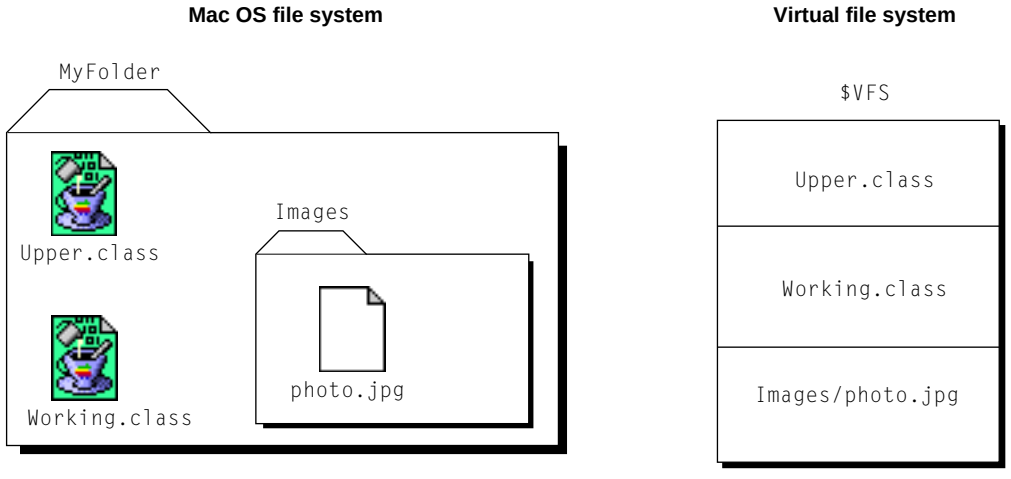
In addition to specifying class paths to files or folders, you can specify a path to a virtual file system (VFS). A virtual file system lets you package Java classes and any other information your Java application requires (such as images or sound clips) in one file. That is, you can store your Java application as one easily portable package.

Items in a virtual file system are stored as though they were in a file hierarchy, so the Java application can access them normally. For example, say you have a Java program that requires two class files, `Upper.class` and `Working.class`, and a JPEG image `photo.jpg` contained in a folder called `Images`. By specifying these in a virtual file system, you can include all of these files in one application file

and preserve the file hierarchy as well. Figure 1-5 compares a virtual file system to a real one.

Note
The contents of a virtual file directory are stored in the data fork of the application file as an uncompressed read-only .zip archive. For more information about data forks, see "Mac OS Resources" (page 26). ◆

Figure 1-5 Real and virtual file systems



To create a virtual file system, you must do the following:

1. Add the root folder of the folder hierarchy to the class path list.
2. Select the folder in the class path list and click the Make VFS button.

For example, in Figure 1-5, the root folder is `MyFolder`. You can add `MyFolder` to the class path list by dragging the folder to the list or by selecting the Add Folder… button. Selecting `MyFolder` in the class path list and clicking the Make VFS Button then makes `MyFolder` the virtual file system.

Note that only one folder may be designated as a virtual file system.

▲ **WARNING**
The folder you designate as the root of your virtual file system should contain only the files you want to package. For example, you should make sure that neither JBindery nor the resulting packaged application is contained within the root folder. Doing so will cause your packaged application to act unpredictably at runtime. ▲

The Expand .zip and .jar Files checkbox specifies whether you want to extract Java classes from .zip and .jar files when packaging them in the application. For example, say you have a .zip file `Airline.zip` that contains `First.class` and `Economy.class`. If you don't specify expanding the .zip file, the packaged application will only contain `Airline.zip`. Otherwise, JBindery extracts the Java classes and places `First.class` and `Economy.class` in the packaged application.

In most cases you should choose to extract your files from .zip and .jar packages. If you do not, however, you must manually add a class path to each unexpanded package using the Add Manually dialog box (as shown in Figure 1-4 (page 14). The class path should be prefixed with `file:///$VFS/`. Select the OK button when finished.

The Properties Panel

Figure 1-6 shows the Properties panel of JBindery. You use this panel to set text properties to be added to the Java environment before calling the application's `main` method. Adding property values using this panel is analogous to calling the method `java.lang.System.setProperties` or specifying `-D`*property=value* on a UNIX command line.

Figure 1-6 The Properties panel

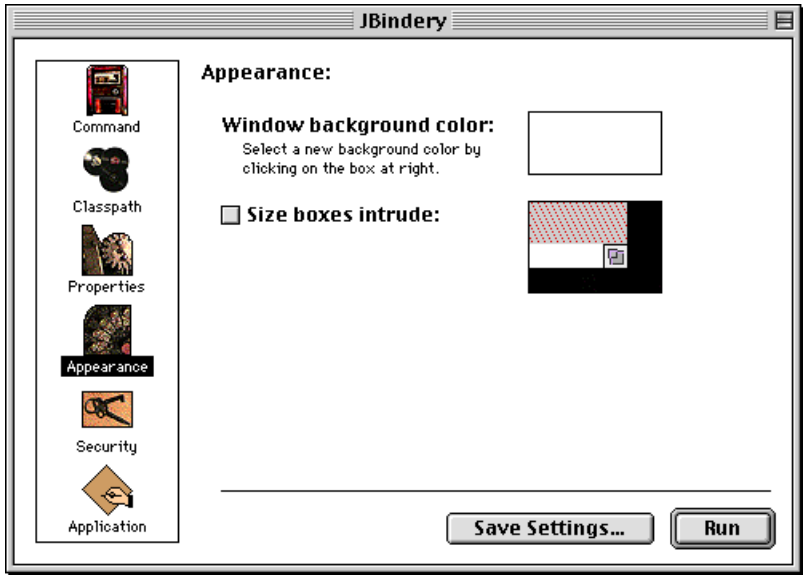
You add a property as a property/value pair. For example, in Figure 1-6, the property `apple` has the value `red`. The Add button adds the property/value pair currently in the text field to the properties list. The Delete button removes the currently selected property from the list.

You can use the Properties panel to override the value of `user.dir`, which indicates the current working directory. For example, you can set the `user.dir` value to `/$APPLICATION/MyStuff`, which indicates the `MyStuff` subdirectory of the application's directory.

The Appearance Panel

Figure 1-7 shows the Appearance panel of JBindery. You use this panel to customize some aspects of the resulting packaged Java application.

Figure 1-7 The Appearance panel

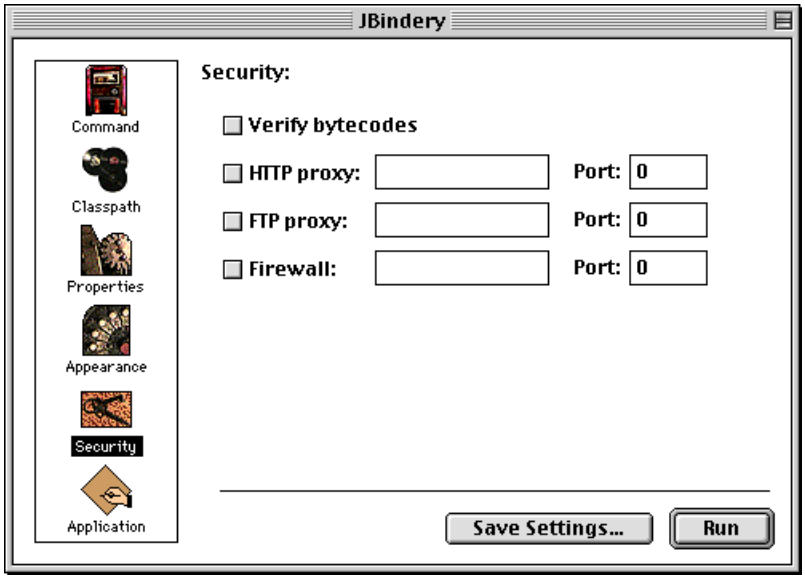


- The Window Background Color field selects the default color for the window background. The Mac OS Window Manager uses this color to erase the window. Clicking within the rectangle brings up a dialog box that lets you choose a background color.
- The Size Boxes Intrude checkbox determines whether the size box should appear within the corner of the actual window. If the size boxes intrude, any Abstract Window Toolkit components that would normally appear under the size box are not drawn. If you do not check this box, an extra strip of empty space is added to the bottom of the window to accomodate the size box.

The Security Panel

Figure 1-8 shows the Security panel of JBindery. You use this panel to specify some security measures for your Java application.

Figure 1-8 The Security panel



- The Verify Bytecodes checkbox specifies whether you want the code verifier to check local Java bytecodes before execution. Checking this box is analogous to selecting the `-verify` option in UNIX. If not checked, JBindery still automatically checks any bytecodes obtained from a remote source (such as over a network). Not checking this box is equivalent to specifying the `-verifyremote` option in UNIX.
- The HTTP Proxy checkbox specifies whether you want to use an HTTP proxy. When checked, the Java application uses the HTTP proxy server you specify (by name and port) whenever accessing an HTTP server.
- The FTP Proxy checkbox specifies whether you want to use an FTP proxy. When checked, the Java application uses the FTP proxy server you specify (by name and port) when making FTP requests.
- The Firewall Proxy checkbox specifies whether you want to use a firewall proxy. When checked, the Java application uses the firewall proxy server specified (by name and port) when accessing servers outside the security firewall.

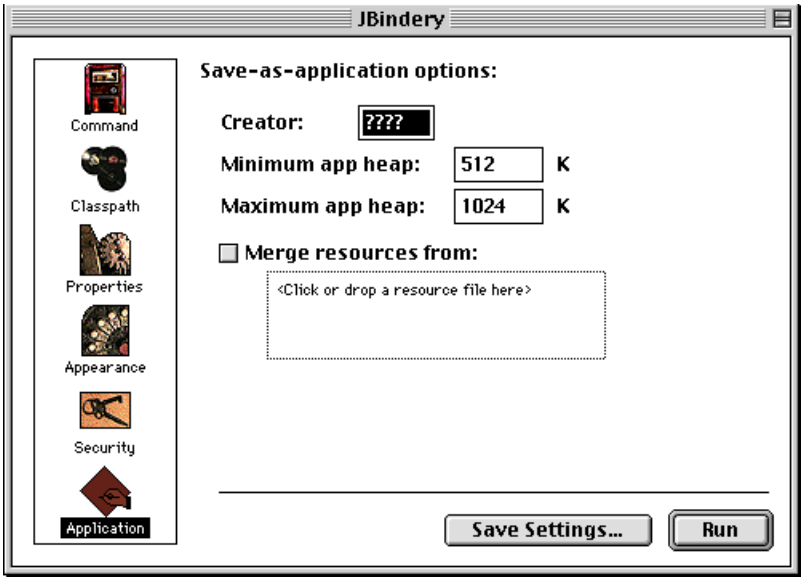
For more information about Java security measures, see the following web site:

http://java.sun.com/security/

The Application Panel

Figure 1-9 shows the Application panel of JBindery. You use this panel to specify some application parameters if you want to package your Java application to run under the Mac OS.

Figure 1-9 The Application panel



- The Creator field lets you specify a creator for your Java application. This unique string identifies the application and any documents that the application may create. If you plan to publically distribute your application, you must register its creator name with Apple through Developer Technical Support to avoid collisions between names used by different developers. You can register a creator online or view currently registered creators at the following Web site:

http://developer.apple.com/dev/cftype/

For more information about creators, see *Inside Macintosh: Macintosh Toolbox Essentials*.

- The Minimum and Maximum App Heap fields let you specify the amount of memory to use when executing this Java program. Mac OS Runtime for Java uses temporary memory for most allocations (the Java virtual machine and so on) so an application heap of 512K is usually sufficient.
- The Merge Resources From checkbox specifies any Mac OS resources you want to add to the packaged Java application. You can drag a compiled resource file onto the box or choose a file by clicking within the box and bringing up a selection dialog box. Resources can contain a custom icon, help text, or other information, but they are optional and do not need to be included with your Java application. For more information, see "Mac OS Resources" (page 26).

Examples

This section contains step-by-step instructions for packaging or executing your Java application using JBindery.

Packaging a Java Application

Packaging a Java application to run under the Mac OS creates a file that contains the following items:

- the Java classes that make up the application
- predefined settings or parameters for the Java application (class paths, arguments, and so on)
- any files the Java application requires (images, data, and so on)
- a wrapper Mac OS application to set up the Java virtual machine and call the application's `main` method
- any Mac OS resources used by the wrapper application

The packaged Java application does not contain the Java virtual machine. The host computer that executes the Java application must supply the virtual machine through the MRJ libraries.

For example, if you wanted to package the `HelloWorld` sample application using JBindery, you can do the following:

1. Drag the class file `HelloWorld.class` onto the JBindery icon. JBindery launches, and `HelloWorld` is listed as the class name in the Command panel.
2. Choose Save Settings... from any of the panels. When the save dialog box appears, name the file `HelloWorld App` and save it. Make sure the Save as Application checkbox is checked.

The resulting file `HelloWorld App` is the packaged Java application. You can double-click on the icon to launch it.

To create a more sophisticated packaged application using a virtual file system, you should observe the following steps:

1. Put all the files your Java application requires into a single folder hierarchy. This folder should not contain any Mac OS resource files.
2. Drag the class file, .zip package, or .jar package containing the application's `main` method onto the JBindery icon. Alternatively, you can launch JBindery and then enter the class name manually in the Command panel.
3. In the Command panel, enter any parameters your application requires and choose paths for console output and input.
4. In the Properties panel, enter any property/value pairs you want to include with your application.
5. In the Appearance panel, select any desired appearance features.
6. In the Security panel, specify any proxy servers, if desired, and choose whether you want to use the code verifier.
7. In the Application panel, choose a signature (also called a creator) for your application. If you plan to distribute your application (whether commercially or as shareware), you should register the creator with Apple. The default creator is `'????'`. See "The Application Panel" (page 20) for more information.
8. If desired, you can change the default application heap sizes in the Application panel.
9. If you have any Mac OS resources you want to include, drag the compiled resource file onto the Merge Resources From box in the Application panel (or click on the box to select the resource file manually). See "Mac OS Resources" (page 26) for more information about resources.

10. If the folder containing your Java application's class files (that is, the top-level folder of your folder hierarchy) does not already appear in the Classpath panel, drag the folder onto the class path list (or click on the Add Folder button to choose the folder using a dialog box).
11. In the class path list, select the folder containing your application's class files and click on the Make VFS button. Doing so designates your folder as the virtual file system. Delete all other entries in the class path list.
12. If you did not choose to expand the .zip or .jar files in your folder hierarchy, you must add paths to each such file by selecting the Add Manually button in the Classpath panel. For example, if you had the file `cookie.jar` in the top level of the folder hierarchy, you must add the path `file:///$VFS/cookie.jar`.
13. If the folder containing your Java application's class files contains any .jar files (expanded or not) and you are running JBindery under MRJ 2.0, you must add class paths to the .jar files using the Add Manually button (as in step 12.).
14. Choose Save Settings... from any of the panels. When the save dialog box appears, choose a name for your packaged Java application file.
15. Select the Save as Application checkbox in the save dialog box and save your packaged file.

During the save process, JBindery displays some progress information as it packages the files you specified in the Virtual File System panel. The saved file can now be launched like a Mac OS application. Alternatively, if you select the Run button while still in JBindery, JBindery automatically quits and launches your packaged application.

If desired, you can keep the Java application class files separate from the application file by not selecting a virtual file system. The resulting application file then contains only the following items:

- predefined settings or parameters for the Java application (class paths, arguments, and so on)
- a wrapper Mac OS application to set up the Java virtual machine and call the application's main method
- any Mac OS resources used by the wrapper application

To build a packaged application with separate class files, you should do the following:

1. Follow steps 2 through 8 as for creating an application with packaged class files.
2. In the Classpath panel, enter any additional paths you want searched when looking for the Java application's class files. Note that paths stored as aliases may not be resolvable if the application is moved to a different computer.
3. Choose Save Settings... from any of the panels. When the save dialog box appears, choose a name for your application file.
4. Select the Save as Application checkbox in the save dialog box and save your packaged file.

The resulting application file can be launched like a Mac OS application assuming that the required Java classes can be found.

Executing a Java Application

You can also use JBindery to simply execute a Java application rather than package it. In such cases, JBindery acts as the wrapper application that sets up the Java virtual machine and calls the main method.

For example, to simply run the sample `HelloWorld` application, you can do the following:

1. Drag the class file `HelloWorld.class` onto the JBindery icon. JBindery launches, and `HelloWorld` is listed as the class name in the Command panel.
2. Select the Run button (or simply hit the Return key).

JBindery then executes the `HelloWorld` application.

To execute a more sophisticated Java application, you should observe the following steps:

1. Drag the class file or .zip package containing the application's main method onto the JBindery icon. Alternatively, you can launch JBindery and then enter the class name manually in the Command panel.
2. In the Command panel, enter any parameters your application requires and choose paths for console output and input.
3. In the Classpath panel, enter the search paths to include when looking for Java classes.
4. In the Properties panel, enter any property/value pairs you want to include with your application.

5. In the Appearance panel, select any desired appearance features.
6. In the Security panel, specify any proxy servers, if desired, and choose whether you want to use the code verifier.
7. In any of the panels, select the Run button (or press the Return key) to execute the Java application.

JBindery quits automatically when you exit the Java application. In cases where you might want to run the Java application repeatedly (for example, for tests during development) you can save a JBindery settings file. To do so, simply select Save Settings... from any of the panels after following steps 1 through 6. When the save dialog box appears, choose a name for the settings file, but do not select the Save as Application checkbox.

IMPORTANT

If you move your settings file to another computer, the Finder may not be able to resolve any class paths stored as aliases. ▲

The saved settings are stored as a JBindery document. Double-clicking on the document icon automatically launches JBindery, which then reads in the settings. To run the Java application with stored settings, you must then select the Run button in any panel.

You can also store frequently used settings by saving them as defaults using the File menu. The default settings (which are stored as a JBindery document in the Preferences folder) are then loaded whenever you launch JBindery directly (that is, when double-clicking or opening the JBindery application itself).

IMPORTANT

The Finder considers JBindery to be the executing application when you use it to simply execute Java applications. For example, if you specified a class path using the variable `/$APPLICATION/`, the path is relative to the location of JBindery, not the location of the Java application's classes. ▲

Mac OS Resources

Under the Mac OS, a file can contain information in the data fork, the resource fork, or both. Traditionally the data fork holds large pieces of contiguous information (as contained in a text document or a flattened QuickTime movie), while the resource fork contains items commonly accessed by Mac OS applications (such as icons or dialog boxes). See *Inside Macintosh: Overview* for a general discussion of Mac OS resources.

You do not need to use resources when building a standalone Java application for the Mac OS. However, if you want to add Mac OS features, you may need to create or modify resources.

You can use the ResEdit resource editor to create or modify resources. You can download ResEdit from the following FTP site:

ftp://ftp.info.apple.com/Apple_Support_Area/Apple.Software.Updates/US/Macintosh/Utilities/

Documentation for ResEdit is available at the following web site:

http://developer.apple.com/techpubs/mac/tools_languages.html

The following sections describe the features you can add to your application using Mac OS resources.

Splash Screens

You can add a splash screen to your packaged application by saving a graphic as a 'PICT' resource with ID -16000. The image is then displayed when your application launches.

Custom Icons

When you build your standalone Java application, JBindery automatically assigns the default application icon (shown in Figure 1-10) to the packaged file.

Figure 1-10 The default application icon

If you prefer a customized icon you must modify or replace the icon resource using ResEdit. See the book *ResEdit Reference* for information about editing icons and adding icon sets with the `'BNDL'` resource.

Note
If the resources you add include a `'BNDL'` resource, the packaged application's bundle bit is automatically set. ◆

IMPORTANT
Apple has strict guidelines for the look of any icons used under the Mac OS. You should consult the book *Macintosh Human Interface Guidelines* (available at the apple.com/developer/ web site) before creating your own custom icons. ▲

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software. Line art was created using Adobe™ Illustrator and Adobe Photoshop.

Text type is Palatino® and display type is Helvetica®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Adobe Letter Gothic.

WRITER
Jun Suzuki

ILLUSTRATOR
Ruth Anderson and Dave Arrigoni

DEVELOPMENTAL EDITOR
Donna S. Lee

PRODUCTION EDITOR
Glen Frank

Special thanks to Karen Wenzel for this revision.

Acknowledgments to Peri Frantz, Rachel Rischpater, and the rest of the MRJ team.