Java 1.4.1 Release Notes

Java

2003-06-11

Contents

Tables and Listings

About This Document

This document provides a high level view of the changes in the Mac OS X implementation of Java from Java 1.3.1 to Java 1.4.1. It does not discuss Java specific changes. That information is fully documented on Sun's website at http://java.sun.com/j2se/1.4/docs/relnotes/features.html. It also highlights key bugs that have been fixed as well as known issues with this release of Java 1.4.1. Additional documentation is available online at http://developer.apple.com/documentation/Java. If you have the Developer Documentation package installed, this documentation is also available in `/Developer/ADC Reference Library/documentation/Java`. Please send documentation feedback to javadevdoc@apple.com.

Filing and Tracking Bugs

If you find issues with the implementation of Java that are not represented in this document or want to follow the resolution of an issue, you may do so online through Radar, Apple's bug tracking system. To access Radar, you need an Apple Developer Connection (ADC) account. You can view the ADC membership options, including the free online membership, at http://developer.apple.com/membership/index.html. With an ADC membership, you can file and view bugs at http://bugreport.apple.com . When filing new bugs for Java on Mac OS X, please use Java (new bugs) for the Component and X as the Version. To file documentation bugs use the Java Documentation (developer) Component.

What's New in Java 1.4.1 in Mac OS X?

The Java 1.4.1 release includes a complete reworking of how Java is implemented for Mac OS X. This release is more than just an update from the Java 1.3.1 classes to the Java 1.4.1 classes. It is more integrated with both the Java Software Development Kit and the core operating system than the Java 1.3.1 implementation. Apple has been able to bring Java 1.4.1 to Mac OS X using about two-thirds less native classes than the 1.3.1 implementation required. With less platform-specific code to maintain, Apple is in a position to more quickly and decisively respond to Java bugs and to provide a cleaner and more integrated Java experience on Mac OS X for you as a developer.

Overview of Changes

Before developing with Java 1.4.1 on Mac OS X, it is important to note a few basic details:

- The Java version is 1.4.1_01.
- The Java 1.4.1 runtime environment requires Mac OS X version 10.2.3 or later.
- Java 1.4.1 is not included as a part of Mac OS X by default. If you are writing Java 1.4.1 applications remember that users will need to download Java 1.4.1 from Apple before they can use your application. They may do this through the Software Update pane of System Preferences or manually from http://www.apple.com/downloads/macosx/. Apple does not provide redistribution licenses for the Java 1.4.1 implementation.
- The Java 1.4.1 Developer Package requires the December 2002 version of the Mac OS X Developer Tools to be installed. Both the Java 1.4.1 Developer Package and the December 2002 Mac OS X Developer Tools are available online at http://connect.apple.com.

Specific Changes From Java 1.3.1 to 1.4.1

The restructuring of the Java implementation has resulted in many changes that are not directly visible to you as a developer. However, there are a few key areas to be aware of:

- This release of Java is installed in addition to the Java 1.3.1 release of Java already included with Mac OS X version 10.2. For the first time, Mac OS X supports two different Java environments, Java 1.3.1 and Java 1.4.1. This change may affect your Java applications deployed on the Mac OS X platform. Chapter 3, "Multiple Versions of Java", (page 11) discusses this issue in more detail.
- With the change from 1.3.1 to 1.4.1 and the underlying architectural changes, the runtime system properties available to you have changed also. These changes are outlined in detail in Chapter 4, "Runtime System Properties", (page 15).
- For Java 1.3.1, AWT and Swing were implemented through the Mac OS X Carbon API. In Java 1.4.1 these implementations have made the transition to the Cocoa API. This change brings a true multithreaded environment and enhancements to the Aqua look and feel. As a Java developer, you don't need to do

anything different to take advantage of these benefits. If though, you rely on native code for any aspects of your Java applications, the change from Carbon to Cocoa will probably affect how you access native code and what native code is most readily available to you. *Java 1.4.1 Development for Mac OS X* provides more details on this topic.

- Browser-based applet support has been improved to provide much smoother and more reliable performance. This includes Java 1.4.1 support from Apple’s Safari browser.
- Native support for the Java Accessibility API is provided by default in AWT. There is no need for you to provide a bridge to native code to make your applications accessible. Users of assistive technologies that have configured the Universal Access system preferences will automatically be able to use your Java application with those technologies. Swing Accessibility is not fully implemented.
- Native support for the security features is provided through the Mac OS X Secure Transport API. This allows users to manage certificates through their Keychain and the Keychain Access application.
- AppleScript support is available for Java applications through two mechanisms. Basic Apple Events are supported through implementing the handlers in `com.apple.eawt.ApplicationListener`. Additional user interface scripting can be done by using the AppleScript UI Element Scripting plug-in available with the December 2002 Mac OS X Developer Tools.
- Jar Bundler, a new application for bundling Java applications as Mac OS X applications, is included. It replaces MRJAppBuilder, which is no longer supported.
- Two versions of the Java Plugin Settings application are now provided. These are installed in `/Applications/Utilities/Java`. One version allows you to access settings for the Java Plug-in 1.3.1 used by Internet Explorer and most other browsers. The other allows you to access settings for the Java 1.4.1 Plug-in used by Safari.
- In Java 1.3.1 in Mac OS X, the ICC_ColorSpace implementation is based on Apple's ColorSync technology. In Java 1.4.1, this is replaced with Sun's implementation. This change fixes compatibility issues when dealing with color profiles in Java on other platforms.
- The `com.apple.mrj` packages have been deprecated. New classes that provide some of the same functionality have been included in the `com.apple.eio` and `com.apple.eawt` packages. These are documented in the *Java 1.4.1 API Reference: Apple Extensions*.
- With Java 1.3.1, you could explicitly specify Java 2D graphics hardware acceleration. With Java 1.4.1, hardware acceleration is handled by the Mac OS X Quartz Extreme graphics engine with no need for developer intervention.
- JDirect is not supported in Java 1.4.1. However, it still works as a part of Java 1.3.1.
- QuickTime for Java is not supported in Java 1.4.1. However, it still works as a part of Java 1.3.1.

Multiple Versions of Java

This release marks a break with tradition for Mac OS X and Java. In previous releases there could only be one version of Java installed on the system. Developers only needed to be concerned with a single Java version for a particular version of Mac OS X. With the Java 1.4.1release, there is the possibility that users will have two different versions of Java on their system. As a developer you need to take this into account for both future development and support of existing Java applications.

Note that there are only two versions of Java when a user has explicitly installed Java 1.4.1. If a user has not explicitly installed Java 1.4.1, they will have only Java 1.3.1, and any 1.4.1-specific code will not work. They can download Java 1.4.1through Software Update or from http://www.apple.com/downloads/macosx/. As a developer, there are many ways you can determine whether Java 1.4.1 is installed, `System.getProperty("java.vm.version")` and `java -version` probably being the simplest. A user shouldn't have to resort to the command line. Instead they can just use the Finder to look in `/Library/Receipts` for an item named `Java1.4.1.pkg`.

The following sections discuss pertinent details you should be aware of to make sure that you are using the correct version of Java for your application. Since there are various ways in which you can deploy Java applications on Mac OS X, the sections are broken down based on your deployment vehicle.The following information applies only if the computer your application is being run on has both Java 1.4.1and Java 1.3.1 installed.

Note: If you are developing WebObjects applications, see the AppleCare Knowledge Base article number 75505 for WebObjects–specific information relevent to using Java 1.3.1 with Java 1.4.1 installed. You can access this information from http://www.info.apple.com/.

Java Applications as Mac OS X Application Bundles

When deploying a Java application on Mac OS X, you should wrap the Java application inside a native Mac OS X application bundle. This application bundle is just a directory with some information about the application itself and a native code stub to launch the Java application. You can construct an application bundle very easily with Jar Bundler which is available in `/Developer/Tools`. For users, the end result is a native application that is double-clickable in the Finder.

By default, existing Java applications that have been bundled as native applications run with the Java 1.3.1 VM. As such, they exhibit the same behavior as they did before installing Java 1.4.1. This behavior provides the cleanest user experience. From a user's perspective, an application continues to work the same way it has always worked. This is important since, from the user's perspective, the application is not necessarily a Java application, but rather a Mac OS X application.

You may determine that you want users to run your existing applications with Java 1.4.1 instead of Java 1.3.1, perhaps to take advantage of new features in the 1.4.1 version. In that case it is your responsibility to designate the appropriate version of Java for your application. You can do this by modifying the `Info.plist` file of the native application bundle.

Note: Even if you do not need to take advantage of Java 1.4.1, you should update your application bundle to note which version of Java you want your application to run with. Although Mac OS X version 10.2 chooses to use Java 1.3.1 if none is specified, future versions of Java or the operating system may choose otherwise.

For new applications, set the VM version in the Build Information pane of Jar Bundler, or the Pure Java-Specific pane in Project Builder's target editor. If you are modifying an existing application, modify the `JVMVersion` property in the `Info.plist` file. This file is in the Contents folder of the application bundle. To view this file in the Finder, Control-click the application icon, then choose "Show Package Contents."

The `Info.plist` file is an XML document, so you can modify it with any text editor. If there is not already a key for JVMVersion, you need to add one. This key should be in the `Java` dictionary. For example, the Java section of the `Info.plist` might look something like Listing 3-1 (page 12).

Listing 3-1 `Info.plist` without `JVMVersion` property

```
<key>Java</key>
    <dict>
        <key>MainClass</key>
        <string>YourApplication</string>
        <key>ClassPath</key>
        <string>$JAVAROOT/YourApplication.jar</string>
    </dict>
```

At the same level as the existing key and string combination for the `ClassPath`, add in a key value of `JVMVersion` and a string value from Table 3-1 (page 12) as appropriate. The resulting Info.plist should look something like that shown in Listing 3-2 (page 12).

Listing 3-2 `Info.plist` with `JVMVersion` property

```
<key>Java</key>
    <dict>
        <key>MainClass</key>
        <string>YourApplication</string>
        <key>JVMVersion</key>
        <string>1.4+</string>
        <key>ClassPath</key>
        <string>$JAVAROOT/YourApplication.jar</string>
    </dict>
```

Table 3-1 Designations for particular versions of Java in Mac OS X version 10.2

Value	Java version used	Notes
1.3.1	1.3.1	Specifies an exact version of Java.
1.3*	1.3.1	Requests the highest version of Java 1.3 available. Note that if Java 1.3 is updated in future releases of Mac OS X the highest version of Java 1.3 will be used.
1.3+	1.4.1	Requests the highest version of Java above 1.3. Note that if Java is updated in future releases of Mac OS X, the highest numbered version will be used.
1.4.1	1.4.1	Specifies an exact version of Java.

Value	Java version used	Notes
1.4*	1.4.1	Specifies the highest version of the Java 1.4 available. Note that if Java 1.4 is updated in future releases of Mac OS X the highest version of Java 1.4 will be used.
1.4+	1.4.1	Specifies the highest version of Java above 1.4. Note that if Java is updated in future releases of Mac OS X, the highest numbered version will be used.

Some older applications may have an `MRJApp.properties` file instead of an `Info.plist` file. For these applications, add the following line with *value* set to an appropriate value from Table 3-1 (page 12): `com.apple.mrj.application.JVMVersion=`*value*.

Java Web Start Applications

Java Web Start applications follow the guidelines laid out for standard JNLP-based applications. Your JNLP file should specify a `J2SE Version` key. This key is evaluated to determine whether to run your application with Java 1.3.1 or Java 1.4.1. The values presented in Table 3-1 (page 12) also apply to Java Web Start applications. For more information on Java Web Start, see the *Developers Guide* at http://java.sun.com/products/javawebstart/1.2/docs/developersguide.html.

If you do not specify a value for the `j2seversion` key, Java 1.3.1 is used by the application.

When a user runs the same Java Web Start application multiple times, Mac OS X provides the option of building a native application bundle around that application. With this release of Java 1.4.1 these application bundles always launch with Java 1.3.1 regardless of what the JNLP file says.

Double-clickable JAR Files

If a JAR file has a `main` class specified in its manifest file, a user can launch it just by double clicking the JAR file. Double-clickable JAR files are run with Java 1.4.1. If you need to run a JAR file with Java 1.3.1, you can either wrap it as a Mac OS X application bundle using Jar Bundler or you can launch it from the command line with `/System/Library/Frameworks/JavaVM.framework/Versions/1.3.1/Commands/java -jar` *YourJARFile.jar*.

Command Line Applications

Running `java` from the command line runs Java 1.4.1. The Java commands in `/usr/bin` are linked to the most current version of Java tools on the system. In this release they are linked to commands in `/System/Library/Frameworks/JavaVM.framework/Versions/1.4.1/Commands`. If you need to run the Java 1.3.1 version of a command line Java tool, you may do so by explicitly calling the appropriate Java 1.3.1 command in `/System/Library/Frameworks/JavaVM.framework/Versions/1.3.1/Commands`.

If you are building shell scripts for distribution to customers that link to a specific version of a Java command in `/System/Library/Frameworks/JavaVM.framework` you should consider wrapping your application as a Mac OS X application with Jar Bundler. Specifying a specific path in the operating system (other than `/usr/bin`) is not the most robust solution since changes to Mac OS X may render these paths obsolete and your application broken.

Java Applets

Java Applets cannot choose which version of Java they invoke. This choice is made instead by the browser. Microsoft's Internet Explorer uses Java 1.3.1. Apple's Safari uses Java 1.4.1. (Applet Launcher also uses Java 1.4.1.)

Native Applications That Invoke Java Through JNI

By default, if the first call into the JavaVM framework is `JNI_CreateJavaVM` where `JNI_VERSION_1_2` specified, the Java 1.3.1 virtual machine (VM) is used. This is done to maintain backward compatibility with the Java 1.3.1 version of applications that may already be in use on the system. If you need to use the 1.4.1 VM and link against the 1.4.1 version of `jni.h`, pass `JNI_VERSION_1_4` instead of `JNI_VERSION_1_2`. If you call `JNI_GetDefaultJavaVMInitArgs` before `JNI_CreateJavaVM` you will also get Java 1.4.1.

Runtime System Properties

Java 1.3.1 on Mac OS X version 10.2 supported some user interface related system properties that are no longer supported in Java 1.4.1. This section highlights those properties as well as the new runtime system properties that are supported.

Unsupported System Properties

The following system properties were available in Java 1.3.1 but are no longer supported in Java 1.4.1. Some have similar replacements as noted.

`com.apple.macos.use-file-dialog-packages`

`com.apple.macos.useScreenMenuBar`
  Use `apple.laf.useScreenMenuBar`.

`com.apple.macos.useSmallTabs`

`com.apple.macosx.AntiAliasedTextOn`
  Use `apple.awt.textantialiasing`.

`com.apple.mrj.application.growbox.intrudes`
  Use `apple.awt.showGrowBox`.

`com.apple.mrj.application.live-resize`

Supported System Properties

The following system properties are supported in Java 1.4.1 in Mac OS X version 10.2. Most of them accept a Boolean value. If they accept a different string that is noted. You can call them from the command line with the `-D` flag to `java` or in your code with `System.setProperty`. For example, you could set your Swing application to use the Mac OS X menu bar with either

```
java -Dapple.laf.useScreenMenuBar="true" yourApplication
```

or

```
System.setProperty("apple.laf.useScreenMenuBar", "true");
```

Integration with the native application environment

`com.apple.mrj.application.apple.menu.about.name`
  Sets the application name that is displayed in the application menu and in the Dock.

  This property take a string of text as an argument. For example, `java -Dcom.apple.mrj.application.apple.menu.about.name="Application Name" yourApplication`.

`apple.laf.useScreenMenuBar`

If you are using the Aqua look and feel, this puts Swing menus in the Mac OS X menu bar. Note that JMenuBars in JDialogs are not moved to the Mac OS X menu bar.

The default value is `false`. Java applications created with Project Builder have this set to `true`.

`apple.awt.showGrowBox`

Most native Mac OS X windows have a resize control in the bottom right corner. By default, Java application windows that use the Aqua look and feel have the functionality of this control, but there is no user interface cue that it is there. This flag causes the resize control to be displayed.

The default value is `false`.

`apple.awt.brushMetalLook`

This flag allows you to display your main windows with the "textured" Aqua window appearance. This should only be applied to the primary application window, and should not affect supporting windows like dialogs or preference windows.

The default value is `false`.

Rendering hints

`apple.awt.antialiasing`

Causes graphic primitives like line, arc, rectangle, and so on to be painted with antialiasing. By default text will also take this setting, though you can override that using `apple.awt.textantialiasing`. Even with this flag set to `true` from the command line, you may still set the `KEY_ANTIALISING` rendering hints for specific objects.

Although this is `false` by default, it is set to `true` when you use the Aqua look and feel. This makes the behavior more consistent with the native Mac OS X user interface. Note that even if you set this to `false` for an application that uses the Aqua look and feel, Aqua user interface elements themselves will still be drawn with antialiasing.

`apple.awt.textantialiasing`

Sets the `KEY_TEXT_ANTIALIASING` rendering hint. Although this inherits the same setting as `apple.awt.antialiasing`, you can override that setting explicitly.

The default value is `false` unless you are using the Aqua look and feel.

`apple.awt.rendering`

Determines whether Graphics2D objects prioritize speed or quality. It sets the `KEY_RENDERING` hint so that it accepts either `VALUE_RENDER_SPEED` or `VALUE_RENDER_QUALITY` as an argument.

`apple.awt.interpolation`

Allows you to set the `KEY_INTERPOLATION` rendering hint to determine which algorithm is used in image transformations. Options include `VALUE_INTERPOLATION_NEAREST_NEIGHBOR`, `VALUE_INTERPOLATION_BILINEAR`, and `VALUE_NTERPOLATION_BICUBIC`.

`apple.awt.fractionalmetrics`

Allows you to set the `KEY_FRACTIONALMETRICS` to use floating point font metrics instead of the default integer metrics. Options include `VALUE_FRACTIONALMETRICS_ON` and`VALUE_FRACTIONALMETRICS_OFF`.

Full screen Java

`apple.awt.fakefullscreen`

This flag causes full screen applications to be displayed in a window. It might be used during development of full screen Java applications.

The default value is `false`.

`apple.awt.fullscreencapturealldisplays`

When you have multiple displays, entering full screen mode normally darkens the secondary screen(s). Setting this to `false` overrides this default behavior and secondary screens are not darkened. This is useful for development purposes like debugging.

The default value is `true`.

`apple.awt.fullscreenhidecursor`

Hides the mouse cursor when in full screen mode.

The default value is `true`.

`apple.awt.fullscreenusefade`

If you change the screen resolution when entering full screen mode, the screen transitions by fading out of the old resolution and back in with the new resolution. If you do not change screen resolution, you normally do not see this fade effect. This setting enables that fade effect regardless of whether you have changed the screen resolution.

The default value is `false`.

Window Positioning

`apple.awt.window.position.forceSafeCreation`

Enforces the creation of new windows on screen. New windows are not created outside of the Desktop where users would not be able to access them.

The default value is `false`.

`apple.awt.window.position.forceSafeProgrammaticPositioning`

Prohibits windows from being moved programatically into a position where users are unable to access them. The `true` setting promotes optimal interaction between the Java environment and the native window server. Setting this to `false` may result in unpredictable behavior in the windowing environment.

The default value is `true`.

`apple.awt.window.position.forceSafeUserPositioning`

Prohibits users from moving windows into a position where they would no longer be able to access them.

The default value is `false`.

Resolved Issues Since Java 1.4.1 Developer Preview 10

The issues discussed here represent some specific bugs that have been fixed since the Java 1.4.1 Developer Preview 10 release. This listing is not a complete list of bugs that have been fixed, but a snapshot of issues that might be of interest. If you still have problems with any of the issues, please file a bug, as described in"Filing and Tracking Bugs" (page 7).

Each issue is broken down as follows:

- A Radar number that Apple uses to the issue is followed by a brief statement of the issue.
- The Description field provides more details about the issue.
- A Resolution or Workaround field tells you what Apple has done to address the issue or suggests actions that you may need to take.

Java AWT

Radar #3169823

Drag and Drop actions result in bus errors.

Description:

If the window is resized after performing drag and drop operations, a segmentation fault or other error is generated.

Resolution:

This has been fixed.

Java Engine

Radar #3165467

Java 1.3.1 code compiled with Java 1.4.1 compiler fails.

Description:

Compiling some Java 1.3.1 code with the 1.4.1 version of javac causes a `java.lang.NoSuchMethodError` when it is run. Java 1.4.1 adds an `append(StringBuffer)` method to the `StringBuffer` class. If you have code that calls `append` with a StringBuffer as an argument, the Java 1.3.1 version of `javac` compiles this to `append(Object)`. The Java 1.4.1 version of `javac` compiles the same code to `append(StringBuffer)`. Running the version compiled with the Java 1.4.1 version of `javac` in a Java 1.3.1 environment returns a `java.lang.NoSuchMethodError`.

Resolution:

To get your code to run on Java 1.3.1, just cast your argument to `java.lang.Object`. For example, `append((Object) stringBufArg)`

Java Events

Radar #3163794

Return key does not generate the expected event.

Description:

The Return key does not work in some applications.

Resolution:

This has been fixed.

Radar #3164000

Copy command does not work.

Description:

When the Copy command is applied to selected text, it does not always copy that text to the pasteboard.

Resolution:

This has been fixed.

Radar #3168246

Cut, Copy, and Paste are not automatically handled in text fields.

Description:

`TextAreas` are not providing built in support for Cut (Command-X), Copy (Command-C), and Paste (Command-V) operations. They should.

Resolution:

This has been fixed so that these features are available by default.

Java HotSpot

Radar #3065092

JBuilder 7 does not launch.

Description:

Once Java 1.4.1 has been installed, JBuilder 7 stops working.

Workaround:

You can modify JBuilder to allow it to launch again under Java 1.3.1. Alternatively, you can launch JBuilder in Java 1.4.1, but only from the command line.

To launch JBuilder 7 in Java 1.3.1:

1. In your JBuilder7 folder, navigate to `JBuilder.framework/bin/`.
2. Open the `jdk.config` file in a text editor. You may need to do this from the shell with the `sudo` command depending on how the permissions to JBuilder7 are set. For example, type `sudo vi jdk.config`.
3. Add the following line: `javapath /System/Library/Frameworks/JavaVM.framework/Versions/1.4.1/Libraries/libjvm_compat.dylib`.
4. Save the file and launch JBuilder7 by double clicking the icon.

To launch JBuilder7 in Java 1.4.1 from the command line, navigate to the `JBuilder.framework/bin` directory inside of the JBuilder7 folder and enter the following command:

```
System/Library/Frameworks/JavaVM.framework/Versions/1.4.1/Commands/java  -
Xbootclasspath/p:../lib/lawt.jar:/System/Library/Java/Extensions/
MRJToolkit.jar -cp ../lib/activation.jar:../lib/ant.jar:../lib/
beandt.jar:../lib/borlandxml.jar:../lib/bsf.jar:../lib/castor-xml.jar:../
lib/castor.jar:../lib/cx.jar:../lib/dbswing.jar:../lib/dbswingdt.jar:../lib/
dbtools.jar:../lib/dt.jar:../lib/dx.jar:../lib/gnuregexp.jar:../lib/
help.jar:../lib/ias_common60.jar:../lib/ias_common65.jar:../lib/
iastools.jar:../lib/internetbeans.jar:../lib/jakarta-regexp-1.1.jar:../lib/
javaws.jar:../lib/jbcl.jar:../lib/jbuilder.jar:../lib/jdbcx.jar:../lib/
jdom.jar:../lib/jds.jar:../lib/jdsremote.jar:../lib/jdsserver.jar:../lib/
junit.jar:../lib/lawt.jar:../lib/linuxDesktopConfigurator.jar:../lib/
mail.jar:../lib/optional.jar:../lib/sqltools.jar:../lib/unittest.jar:../lib/
webserverglue.jar:../lib/xalan.jar:../lib/xerces.jar:../lib/xml4j.jar:../
lib/xmlbeans.jar:../lib/xmldbms.jar:../lib/bes/agentclient.jar:../lib/bes/
asrt.jar:../lib/bes/client.jar:../lib/bes/guicore.jar:../lib/bes/
jaas.jar:../lib/bes/jcert.jar:../lib/bes/jnet.jar:../lib/bes/jsse.jar:../
lib/bes/lm.jar:../lib/bes/vbdev.jar:../lib/bes/vbejb.jar:../lib/bes/
vbjdev.jar:../lib/bes/vbjorb.jar:../lib/bes/vbsec.jar:../lib/bes/xmlrt.jar
com.borland.jbuilder.JBuilder
```

Radar #3123712

Applications that use java.endorsed.dirs generate errors.

Description:

If an application specifies JAR files to load with `java.endorsed.dirs` that application generates an error noting that sharing is disabled.

Resolution:

Using `java.endorsed.dirs` no longer generates this message. Note that using `java.endorsed.dirs` prepends the boot classpath. This means that the shared archive normally used in Mac OS X to decrease memory usage and launch time of Java applications, cannot be used. If you use `java.endorsed.dirs` in an application, that application will not make use of the default shared archive.

Java Sound

Radar #2588625

Audio input doesn't work.

Description:

Audio input is not implemented through JavaSound.

Resolution:

Audio input is implemented through JavaSound, but it is important to note that only 44100 frame rate (in PCM encoding, mono or stereo, 8 or 16 bits per sample) is currently supported for sampled sound input. If you need a different frame rate please resample in your application.

Java Swing

Radar #3174147

JFileChooser.setAccessory does not work.

Description:

Adding accessories to a `JFileChoose` with `setAccessory` does not do anything. The `JFileChooser` just displays as normal.

Resolution:

This has been fixed.

Java Text

Radar #3096892

Font errors in the Console or command line.

Description:

Some of the fonts provided in the underlying operating system have errors. These errors may be displayed when you run your Java application. The errors usually relate to how the font reports its pitch and advance.

Resolution:

These are not Java issues, and there is nothing you should change in your code to correct these errors. The issue is being pursued at the operating system level.

Radar #3183481

Typing ALT (option) + a mnemonic character generates a character instead of giving a field focus.

Description:

Mac OS X uses the Option key to generate some special characters, and as such it cannot be used as an accelerator key on its own.

Workaround:

The accelerator to use with mnemonics is Control-Option.

Known Issues

The issues discussed here are known issues with the Mac OS X implementation of Java 1.4.1. These issues may or may not be resolved in a future release of Java for Mac OS X. Information on tracking these bugs is available in "Filing and Tracking Bugs" (page 7). Additional bug reports on these issues are welcome.

Each issue is broken down as follows:

- A Radar number that Apple uses to track the issue is followed by a brief statement of the issue.
- The Description field provides more details about the issue.
- The Workaround field may give you additional information about how you can work around that particular issue in your code. If it says, "None," there is at the moment no currently a suggested workaround for that particular problem.

Java Aqua LAF

Radar #3156528

JFrames are not garbage collected when using the screen menu bar.

Description:

If the screen menu bar is enabled via `com.apple.laf.useScreenMenuBar`, JFrames with menu bars set with `setJMenuBar` may not be garbage collected. If many such frames are created and then closed, this may result in an `OutOfMemoryError`.

Workaround:

Explicitly remove the JMenuBar when disposing of the frame.

Radar #3166127

JOptionPane dialog buttons have mnemonics.

Description:

`JOptionPane.showConfirmDialog` displays Yes and No with an underscore under the Y and the N. These underscores should not be visible with the Aqua look and feel.

Workaround:

None.

Radar #3167784

JTextArea.setCaretPosition does not place caret correctly.

Description:

After appending text to a `JTextArea` and then calling `setCaretPosition` to set the caret at the end of the appended text, the caret is not always placed at the end of the text.

Workaround:

None.

Java AWT

Radar #3134617

Modal dialogs don't render parent windows inactive.

Description:

With Java 1.3.1 in Mac OS X, a user could not minimize a Java Frame that was behind a modal Java window. In Java 1.4.1, this is possible. This difference arises due to the use of Cocoa for Java windows and is being analyzed as to whether it should be considered a bug.

Workaround:

None.

Java Events

Radar #3164718

Control-drag generates mouseMoved, not mouseDragged.

Description:

Moving the cursor with the Control key and the mouse key held down generates `mouseMoved` events. This should generate `mouseDragged` events.

Workaround:

None.

Java Graphics

Radar #3160445

Apparent random crashes of Java applications.

Description:

Some applications that use Java2D can crash unexpectedly with a crash log that specifies a failure in `Java_apple_awt_CRenderer`. This is a known issue and is being investigated.

Workaround:

None.

Radar #3160678

GradientPaint not working for custom buttons.

Description:

One way to achieve a translucency effect in a component is to use gradient paint with alpha. This does not work in this release. A solid color may be displayed instead.

Workaround:

None.

Radar #3163400

Crash in mutlithreaded draw.

Description:

Multithreaded drawing windows with AWT buttons may result in a crash.

Workaround:

None.

Radar #3174367

Poor graphics performance with some image types.

Description:

32-bit RGB images with alpha can have the alpha channel pre multiplied or not. If your Java2D application makes use of ARGB images where the alpha channel has not been premultiplied, you might notice poor graphics performance in Mac OS X. This is especially evident where the ARGB image is being drawn into as a graphics context.

Workaround:

When constructing ARGB images, designate the image type as ARGB_PRE instead of ARGB.

Java HotSpot

Radar #3129210

The -Xincgc VM flag does not work.

Description:

The train garbage collector is not implemented in Java 1.4.1 so the `-Xincgc` does not have any effect on your code.

Workaround:

None.

Java Printing

Radar #3183076

Java Print Service API does not work.

Description:

The Java Print Service API available in the `javax.print.*` packages is not functional. This should not be confused with the standard Java 2 printing API available in `java.awt.print` which is fully functional.

Workaround:

None.

Java Security

Radar #3173133

Java Kerberos does not work well with the default Mac OS X Kerberos implementation.

Description:

There are known issues with the interaction of Java and the operating system when dealing with Kerberos authentication. The Java environment is unable to correctly locate the credentials cache or tickets on the system.

Workaround:

None.

Java Sun Plugin

Radar #2903948

httpsessions are not maintained for Applets.

Description:

Applets that make multiple HTTP `URLConnections` initiate a new `httpsession` with the server every time. If cookies are used to determine connections, note that cookie data is not neccesarily shared between the browser and the Java Plug-in. This is at the discretion of the browser vender.

Workaround:

Do not make session information dependent on cookies.

Java Swing

Radar #2884768

JMenu.getLocationOnScreen reports incorrect value for screen menu bar.

Description:

When the menu bar is set to be at the top of the screen with `apple.laf.useScreenMenuBar`, `getLocationOnScreen` returns the location of JMenu components as if they were not in the menu bar.

Workaround:

None.

Radar #3168263

javax.swing.JFileChooser.rescanCurrentDirectory doesn't update the file listing.

Description:

If you move files into or out of a directory, `javax.swing.JFileChooser.rescanCurrentDirectory` does not reflect these changes as it should.

Workaround:

None.

Radar #3172089

Resizable windows have no indicator of a resize area.

Description:

By default, Java applications do not have the resize control in the bottom right corner that native Mac OS X applications have. Since many Java developers are not accustomed to having this resize control present when they design their user interface, the decision has been made to not include it by default.

Workaround:

If you do want the resize indicator in your windows, add the `apple.awt.showGrowBox` system property. Set it to `true` either at the command line, with the `-D` flag to `java`, or in your code with `System.setProperty`.

Java Text

Radar #2826318

Double-byte characters are not displayed correctly if the language preference is not set.

Description:

Using logical fonts for screen font or in an editing window, double-byte characters don't display correctly in the default Roman script system.

Workaround:

Change the script behavior setting in the Language pane of the International System Preferences.

Radar #3121780

Many of the fonts available in Mac OS X appear to be broken.

Description:

If you are constructing new fonts based on the value returned from `aFont.getFontName`, where `aFont` is one of the fonts returned from `GraphicsEnvironment.getLocalGraphicsEnvironment().getAllFonts()`, not all of the system fonts in Mac OS X are valid fonts for use in Java.

Workaround:

Use `Font.getName` instead of `Font.getFontName`.

Java Tools

Radar #2962223

Jar Bundler cannot open existing application bundles.

Description:

Jar Bundler is useful for building new application bundles from JAR files or class files. It does not have functionality included for modifying existing application bundles.

Workaround:

None, although you can modify the `Info.plist` settings of an application bundle in any text editor. Most of the Jar Bundler settings are stored in the `Info.plist`.

Radar #3166010

Project Builder uses gdb for Java debugging.

Description:

By default, Project Builder uses `gdb` instead of `jdb` for Java debugging.

Workaround:

In the Target pane select your executable in the Executables list. You should now see a pane that lets you modify the characteristics of that executable. In the Launch Configuration settings change "Configure" to "Debug action." Change "Launch using" to "Java Debugger".

Java Web Start

Radar #2905825

Java Web Start applications may hang for network home directories.

Description:

If a home directory is on an NFS mounted volume, Java Web Start applications hang shortly after they have been launched.

Workaround:

Use either a local account or change the network home directory to a AFP.

Document Revision History

This table describes the changes to *Java 1.4.1 Release Notes*.

Date	Notes
2003-06-11	Multiple VMs chapter updated to include corrected information about the JavaVM key in an Info.plist file as well as some clarification on which version of the Java VM is invoked with JNI. Reference to WebObjects KnowledgeBase article was added.
2003-03-01	Document originally released.