MAC OS RUNTIME FOR JAVA

Programming With MRJToolkit

11/24/98
Technical Publications

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

Simultaneously published in the United States and Canada.

Contents

Tables and Listings

About This Document

MRJToolkit is a set of Java-based classes that you use to access some Mac OS functionality from a Java application. For example, if your Java application allows you to save files, you can use MRJToolkit methods to assign a file type and creator to such a file saved on the Mac OS platform.

You should use MRJToolkit if you are interested in doing any of the following:

- Packaging a Java application using JBindery
- Creating or opening files from the Mac OS Finder for your Java application
- having your application launch or activate when files are dropped onto its desktop icon
- Setting keyboard equivalents for menu items in your Java application

IMPORTANT

MRJToolkit methods are not available for use by Java applets. ▲

How to Use This Document

To understand how to use the MRJToolkit methods to prepare and execute Java applications on the Mac OS platform, you should first read Chapter 1, “Using MRJToolkit,” which gives tutorial information and sample code examples. Chapter 2, “MRJToolkit Reference,” contains descriptions of all the MRJToolkit methods and the required application-defined callback methods. You can reference this chapter while reading Chapter 1 or while writing your code.

Additional Resources

In most cases, extensive knowledge of the Mac OS platform is not necessary when using MRJToolkit methods. However, you should be familiar with

concepts such as file types and creators and have some knowledge of Mac OS menus. If you are not familiar with these concepts, please consult *Inside Macintosh: Macintosh Toolbox Essentials* and *Inside Macintosh: More Macintosh Toolbox* before using MRJToolkit methods. These and other developer resources are available at the following Web site:

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

All of the MRJToolkit methods require that your Java application be packaged using JBindery. JBindery, which does not require any Mac OS programming knowledge, allows you to package a Java application so that you can launch it like any Mac OS application. For information on JBindery, see the document *Using JBindery*.

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

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

This document does not describe the Java language or low-level details of the Java virtual machine. For that information, you should consult JavaSoft documentation, which you can access through the Java home page:

http://java.sun.com/

Conventions Used in This Document

This document uses special conventions to present certain types of information. Words that require special treatment appear in specific fonts or font styles.

Special Fonts

This document uses several typographical conventions.

All code listings, reserved words, command options, resource types, and the names of actual libraries are shown in Letter Gothic `(this is Letter Gothic)`.

Words that appear in **boldface** are key terms or concepts and are defined in the glossary.

Command Syntax

This document uses the following syntax conventions:

`literal`	Letter Gothic text indicates a word that must appear exactly as shown.
italics	Italics indicate a parameter that you must replace with anything that matches the parameter's definition.

Types of Notes

This document uses two types of notes.

Note

A note like this contains information that is useful but that you do not have to read to understand the main text. ◆

IMPORTANT

A note like this contains information that is crucial to understanding the main text. ▲

CHAPTER 1

Using MRJToolkit

Contents

Although most Java code can run unchanged on multiple host platforms, in some cases you might want to access certain platform-specific functions. MRJToolkit allows Java code limited access to the Mac OS platform. Specifically, it allows you to do any of the following from your Java code:

- Set or read special file identifiers (file types and creators) that are used by the Mac OS Finder
- Determine paths to special Mac OS folders, such as the Preferences folder or the Desktop folder
- Respond to simple system events such as requests to open a file, to print, or to quit
- Implement an About box that can be activated from the Apple menu, as Mac OS users expect
- Create keyboard equivalents (shortcut keys) to menu items

Allowing access to the Mac OS platform means that your Java application becomes platform-specific. However, since calls to MRJToolkit do nothing if the appropriate implementations are not available, you can still easily adapt your Java application to work on different platforms.

Although Mac OS programming knowledge is not required to use MRJToolkit, you should read the "Finder Interface" chapter in *Inside Macintosh: Macintosh Toolbox Essentials* for information on how the Finder handles files.

Including MRJToolkit With Your Application

When building your Java application, you must link against the file `MRJToolkitStubs.zip`, which allows your application to access the proper MRJToolkit methods at runtime. This file contains the proper class declarations, but only stub implemenations (that is, the code does nothing).

The Java class files that make up MRJToolkit are stored in the file `MRJClasses.zip`, which is placed in the Extensions folder as part of the normal MRJ installation for MRJ 1.5 and later. If desired, you can choose to package the file `MRJToolkitStubs.zip` with your Java application using JBindery. Doing so ensures that your application can always find a set of MRJToolkit classes (even if they are the stub versions that do nothing). For more information about including Java classes with your application, see the document *Using JBindery*.

Note
MRJToolkit classes are included in `MRJClasses.zip` only with MRJ 1.5 and later. If the MRJToolkit libraries are not available on the host computer at execution time and you have not included `MRJToolkitStubs.zip` with your application, an error occurs. ◆

If you are writing code meant to be compatible with multiple platforms, you can check for the presence of MRJToolkit using the `isMRJToolkitAvailable` method in the class `com.apple.mrj.MRJApplicationUtils`.

Manipulating Files

At times you might want to save files to a Mac OS disk drive from your Java application (such as a text document or a preferences file). While saving files to the Mac OS platform is simple, accessing them afterwards can be more problematic, since files created by a Java application do not normally contain any Mac OS–specific information and therefore cannot be assigned a custom desktop icon. A plain file without a custom icon offers the user no clues as to what application created the file or what its contents may be. Attempts to open such a file from the Mac OS Finder (the system application that manages files and the desktop) may not work, since the Finder does not know which application it should use to open it.

To solve this problem, MRJToolkit allows you to assign special Mac OS identifiers to files saved from Java applications. You can then open or search for such files just as you could from a Mac OS application. MRJToolkit also allows you to search for specific applications or special folders (such as the Preferences folder).

File Types and Creators

To identify and handle files properly, the Mac OS Finder requires two 4-byte identifiers: the file type and the creator. A **file type** is a string that specifies the contents of a file. For example, the file type `'APPL'` identifies the file as an application and therefore executable. A file type of `'TEXT'` means that the file contains raw text. Any application that can read raw text can open a file of type `'TEXT'`.

However, many applications create files that contain application-specific information. For example, a word processor document might contain formatting and style information in addition to the raw text. Since only the application that created it can make use of the information, the application often assigns the file a proprietary file type.

To identify the application that created a document, the Finder relies on a string called a **creator**. For example, the SimpleText application (the default text editor installed with Mac OS system software) assigns the creator `'ttxt'` to all its documents. If you double-click on a document that has the `'ttxt'` creator, the Finder knows to look for the SimpleText application to open it (the application also bears the creator `'ttxt'`). The Finder also uses the creator to assign the "correct" icon to a file so that users can tell what application created it. Creators also allow the Finder to provide useful information about a file when you select the Get Info item in the File menu.

Note
Creators may not necessarily indicate the actual creator of a file, but rather what application should open it. For example, if you use an editor to create an HTML document, you might want to assign a browser's creator of the file rather than the HTML editor's creator. Double-clicking on the document then opens the appropriate browser rather than the HTML editor. ◆

The MRJToolkit class `com.apple.mrj.MRJFileUtils` contains a number of methods that you can use to set the file type and creator of a file.

If you plan to publicly distribute your application, you must register its creator and any proprietary file types 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://devworld.apple.com/dev/cftype/main.html

For more detailed information about how the Finder handles file types and creators, see the "Finder Interface" chapter in *Inside Macintosh: Macintosh Toolbox Essentials*.

The MRJOSType Data Type

The Mac OS designates the data type `OSType` to hold file types, creators, or the name of a folder. MRJToolkit allows access to `OSType` data through a wrapper object of type `MRJOSType`. You manipulate an MRJ OS type in the Java environment just as you would a Mac OS type in the Mac OS environment.

Setting the Default File Type and Creator

When packaging your Java application JBindery assigns the default creator you specify (or `'????'` if you do not choose one) and assumes that all files created by the application will have the file type `'TEXT'`. To override these settings, you use the `setDefaultFileType` and `setDefaultCreator` methods. Any files that your application creates will then automatically have these default values set. Listing 1-1 sets the default creator and file type and creates a file on the Mac OS.

Listing 1-1 Setting the default file type and creator

```
import com.apple.mrj.*;
import java.io.*;

...

public static void testFileUtils() {
        /* first, set the current file and creator */
        MRJOSType newType = new MRJOSType("TEXT");
        MRJOSType newCreator = new MRJOSType("ttxt");

        MRJFileUtils.setDefaultFileType(newType);
        MRJFileUtils.setDefaultFileCreator(newCreator);

        /* create a File object with the current file and creator */
        File f = new File("TestFile");

        if (f.exists())
            f.delete();

        try {
            PrintStream ps = new PrintStream(new FileOutputStream(f));
```

```
            ps.println("Hello, World");
            ps.close();
            }
        catch (IOException e) {
            fail("Failed to write output file", e);
            return;
            }
```

The created file `TestFile` has the file type `'TEXT'` and the creator `'ttxt'`, indicating that it is a text file and should be opened using SimpleText. Since no path was specified, `TestFile` appears in the default directory (typically the application's directory).

Setting or Reading File Types and Creators for Existing Files

Sometimes you may want to determine or set a file type or creator for a file that already exists. For example, if the user requests that a file be opened, the application might check the file type to make sure it can handle that type of file. You can read the file type and creator using the `getFileType` and `getFileCreator` methods respectively. To set the file type and creator, you can use either the `setFileType` and `setFileCreator` methods or the `setFileTypeAndCreator` method. The code fragment in Listing 1-2 shows an example of reading and setting the file type and creator.

Listing 1-2 Reading and setting the file type and creator

```
try {
        MRJOSType curType = MRJFileUtils.getFileType(f);
        MRJOSType curCreator = MRJFileUtils.getFileCreator(f);

        // make sure they're what we expect
        if (! (curType.equals(new MRJOSType ("TEXT") &&
                curCreator.equals(new MRJOSType ("ttxt"))))
            throw new IOException("Unexpected file type or creator");
    } catch (IOException e) {
        fail("Couldn't get file type or creator", e);
        return;
        }
```

```
try {
    MRJFileUtils.setFileTypeAndCreator(f, new MRJOSType("TEXT"),
            new MRJOSType("CWIE"));
    } catch (Exception e) {
        fail("Can't set file type and creator", e);
        return;
        }
```

This example checks to see if the file in question is a SimpleText text file (file type `'TEXT'`, creator `'ttxt'`). If so, it then changes the file type and creator to be that of a CodeWarrior text file. Note that since the file type is not changed, you could have called the `setFileCreator` method to just set the creator to `'CWIE'`. After changing the creator, double-clicking on the file causes the file to be opened using CodeWarrior rather than SimpleText.

Finding an Application With a Given Creator

If you want to find an application with a given creator, you can call the `findApplication` method. For example, if you have an HTML file with a given browser's creator, you can search for the browser and then open the file with it. The code fragment in Listing 1-3 searches for the SimpleText application and then launches it by passing the file to `Runtime.runtime.exec`. (In a similar manner, you could search for a browser and have it open a local HTML file.)

Listing 1-3 Finding the path to an application

```
try {
        File cw = MRJFileUtils.findApplication(new MRJOSType("ttxt"));

        // launch SimpleText with our new text file
        String params[] = { cw.toString(), f.toString() };

        // parameters to runtime exec:
        // { AppToRun, FileToOpen, ... }
        Runtime.getRuntime().exec(params);

    } catch (Exception e) {
```

```
        fail("Can't find SimpleText", e);
        return;
        }
```

The `findApplication` method searches the Mac OS desktop database looking for an application with the correct creator. If more than one copy of the application exists, `findApplication` may find a copy different from the one you expect, depending on the state of the desktop database.

Finding Special Folders

The Mac OS platform has numerous special folders (that is, directories) that contain specialized files. For example, system extensions are stored in the Extensions folder, while fonts are stored in the Fonts folder. If your Java application accesses or saves files on the Mac OS platform, you may need to find the path to a special folder. For example, if you want to save application default settings, you should store these in the Preferences folder.

To find the path to a particular folder, you must call the `findFolder` method while specifying the folder you want to locate. Listing 1-4 shows a code fragment that locates the Preferences folder and prints out the path to standard output.

Listing 1-4 Finding the Preferences folder

```
try {
        System.out.println("Preferences folder is: " +
            MRJFileUtils.findFolder(
            MRJFileUtils.kPreferencesFolderType));
        }
catch (FileNotFoundException e) {
        fail("Couldn't get Preferences folder", e);
        return;
        }
```

The `kPreferencesFolderType` constant specifies that you are looking for the Preferences folder. See “Special Folder Constants” (page 32) for a complete listing of possible folders and their constants.

Responding to Simple System Events

On the Mac OS platform, applications can respond to simple events passed to it by the Finder or other applications. For example, if the user drops a file onto an application's icon, the Finder launches the application (if not already open) and passes it an "open file" event instructing it to open the file if possible.

Note
Events on the Mac OS platform are handled through Apple Events, which are described in detail in *Inside Macintosh: Interapplication Communication.* No knowlege of Apple Events is needed when using MRJToolkit. ◆

MRJToolkit allows your Java application to respond to the following system events:

- requests to open a document
- requests to print a document
- requests to quit

MRJToolkit also allows your application to respond to an "About box selected" event when the About item in the Apple menu is selected. An About box is a window that gives information about the application, such as the version number, serial number, and credits (members of the development team and so on).

To make your application aware of the event, you must implement an interface that handles the particular event and then register the handler method with MRJToolkit. For example, to create a quit handler, you must implement the interface `MRJQuitHandler`. This interface has the form shown in Listing 1-5.

Listing 1-5 The MRJQuitHandler interface

```
package.com.apple.mrj

public interface MRJQuitHandler {
    public void handleQuit();
}
```

To use this interface, the class that implements it *must* contain a method named `handleQuit` that defines the actions to take when the application receives a quit request. In addition, the application must also register the method name with MRJToolkit by calling the appropriate registration method in the class `com.apple.mrj.MRJApplicationUtils`. For example, to register the quit event handler, you must call the `MRJRegisterQuitHandler` method. Listing 1-6 shows an example of implementing a quit handler.

Listing 1-6 Implementing the quit event handler

```
import com.apple.mrj.*;
import java.io.*;

...

class QuitTest implements MRJQuitHandler {

    public QuitTest() {
        System.out.println("Select Quit from the Apple menu or shutdown
                    to test event handler.");

        MRJApplicationUtils.registerQuitHandler(this);
    }

    public void handleQuit() {
        System.exit(0); /* Quit the MRJ runtime */
        }
}
```

Since the call to `registerQuitHandler` occurs within the class that contains `handleQuit`, it can use the `this` variable to reference the handler.

Table 1-1 shows the available handler interfaces, their corresponding registration methods, and the conditions under which an event is sent. All the

registration methods are members of the class `com.apple.mrj.MRJApplicationUtils`.

Table 1-1 Event handling methods

Interface	Handler Name	Registration Method	When Activated
`MRJOpenDocumentHandler`	`handleOpenFile`	`registerOpenDocumentHandler`	When the Finder requests that a file be opened (for example, when the user drags an appropriate file onto the application icon).
`MRJQuitHandler`	`handleQuit`	`registerQuitHandler`	When the application receives a request to quit (for example, before shutting down, or if the user selected the default Quit Item in the Apple Menu).
`MRJPrintDocumentHandler`	`handlePrintFile`	`registerPrintDocumentHandler`	When the Finder requests that the application print (for example, if the user selects a file in the Finder and chooses the Print menu item).
`MRJAboutHandler`	`handleAbout`	`registerAboutHandler`	When the user selects the About item in the application's Apple Menu.

Assigning Keyboard Equivalents to Menu Items

Many Mac OS applications allow the user to select a menu item by entering a special key combination, such as the Command key followed by a letter or number. Such combinations are called **keyboard equivalents** (or, sometimes, *shortcut keys*). For example, many Mac OS applications have assigned the keyboard equivalent Command-o to the Open File menu item. Entering this key combination functions exactly as if the user selected the Open File menu item with the mouse.

If you want to assign keyboard equivalents to menu items in your Java application, you can do so using the `SetMenuItemCmdKey` method in the class `com.apple.mrj.MRJMenuUtils`.

IMPORTANT

Future versions of MRJ, which implement Sun's JDK 1.1 standards, will let you assign keyboard equivalents by binding `MenuAccelerator` objects to a menu item, so you do not need to use `SetMenuItemCmdKey`. ▲

The `SetMenuItemCmdKey` method allows you to assign a keyboard equivalent of the form "Command-*character*" to a menu item. The `SetMenuItemCmdKey` method is overloaded; you can specify the menu item by referencing the menu item by name or by menu name and menu index. Listing 1-7 shows code that creates items in a menu and assigns them keyboard equivalents.

Listing 1-7 Assigning a keyboard equivalent to a menu item.

```
import com.apple.mrj.*;
import java.io.*;
import java.awt.*;

...

class MenuTest extends Frame {
    MenuTest() {
        MenuBar mb = new MenuBar();
        Menu menu = new Menu("Menu");
```

```
        menu.add("Hop");
        menu.add("Skip");
        menu.add("Jump");
        mb.add(menu);
        setMenuBar(mb);

        MRJMenuUtils.setMenuItemCmdKey(menu, 0, '1');
        MRJMenuUtils.setMenuItemCmdKey(menu, 1, '2');
        MRJMenuUtils.setMenuItemCmdKey(menu, 2, '3');

        show();
    }

    public boolean handleEvent(Event eve) {
        if (eve.id == Event.ACTION_EVENT)
            System.out.println(eve);
        return false;
    }
}
```

This code creates a menu named `Menu` which has three menu items associated with it: `Hop`, `Skip`, and `Jump`. It then assigns each a keyboard equivalent. For example, entering Command-1 would select the menu item `Hop`. The keyboard equivalent appears in the menu item next to the item name.

The rest of the code is a simple event handler that traps an event (that is, the selection of a menu item) and prints the contents of the Apple Event record to the standard output.

Although you can choose any letter or number to go with the Command key, most Mac OS applications have certain keyboard equivalent conventions, which you should follow. Table 1-2 shows some of the more common conventions.

Table 1-2 Some Mac OS keyboard equivalent conventions

Keyboard Equivalent	Menu Action
Command-N	Create a new document
Command-O	Open a file
Command-W	Close a window
Command-Q	Quit the application
Command-P	Print the current document
Command-S	Save the current document
Command-A	Select all
Command-C	Copy selection
Command-X	Cut selection
Command-V	Paste into selection
Command-Z	Undo last action

For more information about keyboard equivalents, see the "Menu Manager" chapter of *Inside Macintosh: Macintosh Toolbox Essentials*.

CHAPTER 2

MRJToolkit Reference

Contents

This chapter describes all the MRJToolkit classes, constants, and methods.

All the MRJToolkit Java classes are stored in the package `com.apple.mrj`, which is part of the `MRJClasses.zip` file distributed with Mac OS Runtime for Java. For development, your application should import the package `MRJToolkitStubs.zip` to be able to access the MRJToolkit classes.

The MRJOSType Class

Description

The following methods belong to the class `com.apple.mrj.MRJOSType`. These methods allow you to handle a special type `MRJOSType`, which acts as a Mac OS type in the Java runtime environment.

Methods

MRJOSType

Converts a string, value, or byte array to a value of type `MRJOSType`.

```
public MRJOSType(String fromString);

public MRJOSType(int fromInt);

public MRJOSType(byte fromBytes[]);
```

`fromString` The string to convert to type `MRJOSType`. This string must be four characters in length. Since each character is represented by 2 bytes in the Unicode standard, this method uses only the low-order byte of each character in `fromString`.

`fromInt` The integer value to convert to type `MRJOSType`.

`fromBytes`	The array of bytes to convert to type `MRJOSType`. The array must contain 4 bytes.

DISCUSSION

This overloaded method defines a wrapper object that acts as the equivalent of a Mac OS 4-byte character constant (such as a creator, file type, or special folder name). When specifying a Mac OS creator, file type, or special folder name in the Java environment, you must refer to it using an MRJ OS type.

SEE ALSO

The "Finder Interface" chapter in *Inside Macintosh: Macintosh Toolbox Essentials* for information about file types, creators, and folder names.

equals

Compares a value to one of type `MRJOSType` or an integer.

```
public final boolean equals (MRJOSType type);

public final boolean equals (int type);
```

`type`	The value to compare against.
method result	True if the value matches what you compared it against, false otherwise.

DISCUSSION

For example, if you received an MRJ OS type in the variable `myType`, then

```
myType.equals(new MRJOSType ("TEXT")) ;
```

is true if the MRJ OS type is of type `'TEXT'`.

toInt

Converts a value of type `MRJOSType` to an integer.

```
public int toInt();
```

method result The converted integer value.

DISCUSSION

`myVal.toInt()` is the value `myVal` converted to an integer.

toString

Converts a value of type `MRJOSType` to a string.

```
public String toString();
```

method result The converted string.

DISCUSSION

`myVal.toString()` is the value `myVal` converted to a string.

The MRJFileUtils Class

Description

The following constants and static methods belong to the class `com.apple.mrj.MRJFileUtils`. You use these methods to set or modify parameters for a Mac OS file and to locate special folders (such as the System Folder) from your Java application. For example, if your Java program saves a file to disk, you can set the Mac OS creator and the file type. Doing so makes it possible to launch the Java application by double-clicking on the associated file.

Special Folder Constants

When searching for the path for special folders, you must specify the folder that you are looking for using the following constants.

```
public static final MRJOSType
    kSystemFolderType = newMRJOSType(0x6D616373),              // 'macs'
    kDesktopFolderType = newMRJOSType(0x6465736B),             // 'desk'
    kTrashFolderType = newMRJOSType(0x74727368),               // 'trsh'
    kWhereToEmptyTrashFolderType = new MRJOSType(0x656D7074),  // 'empt'
    kPrintMonitorDocsFolderType = new MRJOSType(0x70726E74),   // 'prnt'
    kStartupFolderType = new MRJOSType(0x73747274),            // 'strt'
    kShutdownFolderType = new MRJOSType(0x73686466),           // 'shdf'
    kAppleMenuFolderType = new MRJOSType(0x616D6E75),          // 'amnu'
    kControlPanelFolderType = new MRJOSType(0x6374726C),       // 'ctrl'
    kExtensionFolderType = new MRJOSType(0x6578746E),          // 'extn'
    kFontsFolderType = new MRJOSType(0x666F6E74),              // 'font'
    kPreferencesFolderType = new MRJOSType(0x70726566),        // 'pref'
    kTemporaryFolderType = new MRJOSType(0x74656D70);          // 'temp'
```

Constant Descriptions

`kSystemFolderType`
: The System Folder.

`kDesktopFolderType`
: The Desktop folder.

`kTrashFolderType`
: The Trash folder (for single-user systems).

`kWhereToEmptyTrashFolderType`
: The shared Trash folder for networked users.

`kPrintMonitorDocsFolderType`
: The PrintMonitor folder.

`kStartupFolderType`
: The Startup Items folder.

`kShutdownFolderType`
: The Shutdown Items folder.

`kAppleMenuFolderType`
: The Apple Menu folder.

`kControlPanelFolderType`
: The Control Panels folder.

`kExtensionFolderType`
: The Extensions folder.

`kFontsFolderType`
: The Fonts folder.

`kPreferencesFolderType`
: The Preferences folder.

`kTemporaryFolderType`
: The Temporary Items folder. This folder is invisible to the user.

Some of the folders may be absent due to differences in system software versions or system configuration.

Methods

setDefaultFileType

Sets the default file type for the Java application.

```
public static void setDefaultFileType (
                    MRJOSType defaultType);
```

`defaultType` The 4-byte file type you wish to assign as the default.

DISCUSSION

On the Mac OS, a file type is a 4-byte character constant that identifies the type of file to the Finder (for example, `'TEXT'` for a text file, or `'APPL'` for an application). If you do not specify a default file type, any saved files will have the file type `'TEXT'`.

SEE ALSO

The “Finder Interface” chapter in *Inside Macintosh: Macintosh Toolbox Essentials* for information about file types.

setDefaultFileCreator

Sets the default creator for the Java application.

```
public static void setDefaultFileCreator (
                    MRJOSType defaultCreator);
```

`defaultCreator`
The 4-byte creator you wish to assign as the default.

DISCUSSION

On the Mac OS platform, the creator is a 4-byte character constant that the Finder uses to identify the application that should open a document. The application itself has the same creator as its documents. If you do not specify a default creator, any saved files will have the current application's creator.

SEE ALSO

The "Finder Interface" chapter in *Inside Macintosh: Macintosh Toolbox Essentials* for information about creators.

setFileTypeAndCreator

Sets the file type and creator of an existing Mac OS file.

```
public static final void setFileTypeAndCreator (
                    File file,
                    MRJOSType type,
                    MRJOSType creator)
                    throws IOException;
```

`file` The file whose file type and creator you want to set.

`type` The 4-byte file type you want to assign.

`creator` The 4-byte creator you wish to assign.

DISCUSSION

On the Mac OS, a file type is a 4-byte character constant that identifies the type of file to the Finder (for example, `'TEXT'` for a text file, or `'APPL'` for an application). A creator is a 4-byte character constant that the Finder uses to identify the application that created a document; doing so allows the Finder to launch or activate the appropriate application when the document is opened.

SEE ALSO

The `setFileType` method (page 35).

The `setFileCreator` method (page 36).

The "Finder Interface" chapter in *Inside Macintosh: Macintosh Toolbox Essentials* for information about file types and creators.

setFileType

Sets the file type for an existing Mac OS file.

```
public static final void setFileType (
                    File file,
                    MRJOSType type)
                    throws IOException;
```

`file` The file whose file type you want to set.

`type` The 4-byte file type you wish to assign.

DISCUSSION

On the Mac OS, a file type is a 4-byte character constant that identifies the type of file to the Finder (for example, `'TEXT'` for a text file, or `'APPL'` for an application). This method throws `IOException` if the desired file cannot be found.

SEE ALSO

The `setFileTypeAndCreator` method (page 34).

The `setFileCreator` method (page 36).

The "Finder Interface" chapter in *Inside Macintosh: Macintosh Toolbox Essentials* for information about file types.

setFileCreator

Sets the creator for an existing Mac OS file.

```
public static final void setFileCreator (
                    File file,
                    MRJOSType creator)
                    throws IOException;
```

`file`	The file whose creator you want to set.
`creator`	The 4-byte creator you wish to assign as the default.

DISCUSSION

On the Mac OS, a creator is a 4-byte character constant that the Finder uses to identify the application that should be used to open the document file. This method throws `IOException` if the desired file cannot be found.

SEE ALSO

The `setFileTypeAndCreator` method (page 34).

The `setFileType` method (page 35).

The "Finder Interface" chapter in *Inside Macintosh: Macintosh Toolbox Essentials* for information about creators.

getFileType

Gets the file type of an existing Mac OS file.

```
public static final MRJOSType getFileType (
                    File file)
                    throws IOException;
```

`file`	The file whose file type you want to obtain.
method result	The 4-byte file type of the file.

DISCUSSION

On the Mac OS platform, a file type is a 4-byte character constant that identifies the type of file to the Finder (for example, `'TEXT'` for a text file, or `'APPL'` for an application). This method throws `IOException` if the desired file cannot be found.

SEE ALSO

The `setFileTypeAndCreator` method (page 34).

The `setFileType` method (page 35).

The "Finder Interface" chapter in *Inside Macintosh: Macintosh Toolbox Essentials* for information about file types.

getFileCreator

Gets the creator of an existing Mac OS file.

```
public static final MRJOSType getFileCreator (
                    File file)
                    throws IOException;
```

`file`	The file whose creator you want to obtain.
method result	The 4-byte creator of the file.

DISCUSSION

On the Mac OS, a creator is a 4-byte character constant that the Finder uses to identify the application that should open the document file. If the desired file cannot be found, this method throws `IOException`.

SEE ALSO

The `setFileTypeAndCreator` method (page 34).

The `setFileCreator` method (page 36).

The "Finder Interface" chapter in *Inside Macintosh: Macintosh Toolbox Essentials* for information about creators.

findFolder

Returns the path to a special Mac OS folder.

```
public static File findFolder (MRJOSType
                    folderType)
                    throws FileNotFoundException;
```

`folderType`	The folder you are looking for. For the list of constants you can specify in this parameter, see "Special Folder Constants" (page 32).
method result	The file object that references the folder you specified.

DISCUSSION

The Mac OS has several special folders that applications often need to access (for example, the Preferences folder); this method searches the startup volume for the desired folder and returns the path. If the folder cannot be found, the method throws `FileNotFoundException`.

SEE ALSO

The "File Manager" chapter in *Inside Macintosh: Files.*

findApplication

Returns the path to an application.

```
public static File findApplication (
                    MRJOSType applSig)
                    throws FileNotFoundException;
```

`applSig` The 4-byte creator of the application you are looking for.

method result The file object that references the application you specified.

DISCUSSION

This method searches all local disks for the application. The search algorithm is identical to that used by the Finder. If the application cannot be found, the method throws `FileNotFoundException`.

SEE ALSO

The "Finder Interface" chapter in *Inside Macintosh: Macintosh Toolbox Essentials* for information about search paths and creators.

The MRJApplicationUtils Class

Description

The following static methods belong to the class `com.apple.mrj.MRJApplicationUtils`. You use these methods to check for the presence of MRJToolkit or to register handlers for high-level user events, such as printing or selecting the About box. See "Application-Defined Methods" (page 44) for more information about the form required for the event handlers.

IMPORTANT

You can use these methods only with Java applications packaged with JBindery. ▲

Methods

isMRJToolkitAvailable

Checks for the presence of MRJToolkit.

```
public static final boolean isMRJToolkitAvailable();
```

method result True if MRJToolkit is present on the host platform, false otherwise.

DISCUSSION

You can include this method only if you are building your application with the version of MRJToolkit included with the MRJ 2.0 SDK or later. However, it will still return the correct value (true) when called on host platforms running MRJ 1.5.

registerAboutHandler

Registers the handler method to be called when the About menu item is selected.

```
public static final void registerAboutHandler (
                    MRJAboutHandler handler);
```

`handler` The name of the object that implements the `handleAbout` method.

DISCUSSION

You must have defined the handler method before calling this method. If you do not register the handler method, selecting the About menu item does nothing.

The default About menu item included in the Apple menu, while usable, should be replaced to conform to Apple's human interface guidelines; you do

so by defining a new menu (in a resource of type `'MENU'` with ID 1128) when you package your application.

SEE ALSO

The `handleAbout` method (page 44).

registerOpenDocumentHandler

Registers the handler method to be called when the application receives a request to open a document.

```
public static final void registerOpenDocumentHandler (
                    MRJOpenDocumentHandler handler);
```

`handler` The name of the method that handles the file drop event.

DISCUSSION

You must have defined the handler before calling this method.

SEE ALSO

The `handleOpenFile` method (page 45).

registerPrintDocumentHandler

Registers the handler method to be called when the user requests that a document be printed from the Finder.

```
public static final void registerPrintDocumentHandler (
                    MRJPrintDocumentHandler handler);
```

`handler` The name of the method that handles the print request.

DISCUSSION

You may not be able to fully implement a print handler method due to lack of printing support in JDK 1.0.2 or earlier. You must have defined the handler method before calling this method.

SEE ALSO

The `myHandlePrintFile` method (page 46).

registerQuitHandler

Registers the handler method to be called when the application receives a request to quit.

```
public static final void registerQuitHandler (
                    MRJQuitHandler handler);
```

`handler` The name of the method that handles the quit request.

DISCUSSION

You must have defined the handler method before calling this method.

Java applications packaged with JBindery have a default Quit selection in the Apple Menu. However, to conform to Apple's human interface guidelines, you should replace the default Apple Menu (by defining a new menu in a resource of type `'MENU'` with ID 1128 when you package your application) and include a Quit selection in one of the Java-based menus (typically the File menu).

SEE ALSO

The `handleQuit` method (page 47).

The MRJMenuUtils Class

Description

The following overloaded method belongs to the class `com.apple.mrj.MRJMenuUtils`. You use this method to assign keyboard equivalents to Java menu items (that is, `java.awt.MenuItem` objects). You can specify the menu item by name or as an indexed member of a menu.

Methods

setMenuItemCmdKey

Sets a key as a keyboard equivalent for a menu item.

```
public static final void setMenuItemCmdKey (Menu menu,
                    int itemIndex, char ch);

public static final void setMenuItemCmdKey (MenuItem item, char ch);
```

`menu`	The name of the menu that contains the desired menu item.
`itemindex`	The index number of the menu item. This index is zero-based.
`ch`	The character to set as the keyboard equivalent. The menu item is activated by selecting Command + `ch`.
`item`	The name of the desired menu item.

DISCUSSION

You can specify the menu item by its name or by indicating the menu that contains it and its index number (the first item in the menu has index 0, the

second has index 1, and so on). You can assign only one character to act as the keyboard equivalent (for example, “A” but not “option- A”). For example, if you assign G as the character, Command-G selects the menu item.

SEE ALSO

The “Menu Manager” chapter in *Inside Macintosh: Macintosh Toolbox Essentials.*

Application-Defined Methods

The following public interfaces contain methods that you must define yourself. These methods handle various user events such as selecting the About box, quitting the application, and so on. You can use these interfaces only with Java applications packaged with JBindery.

IMPORTANT

The names of the methods described are fixed. For example, if you implement the `MRJAboutHandler` interface, you *must* include a method with the name `handleAbout`. ▲

The MRJAboutHandler Interface

The `com.apple.mrj.MRJAboutHandler` interface contains one method, `handleAbout`, which is called when the user selects the About menu item.

handleAbout

Performs an action when the user selects the About menu item.

```
public void handleAbout ();
```

DISCUSSION

This application-defined method is called when the user selects the About menu item in the Apple menu. Typically this method displays a splash screen containing information about the application, such as a brief description, copyright information, the names of the development team, and so on.You must register the method name by calling the `registerAboutHandler` method (page 40) when initializing the application. The `handleAbout` method is only useful if you have packaged your Java application using JBindery.

SEE ALSO

The `registerAboutHandler` method (page 40).

The MRJOpenDocumentHandler Interface

The `com.apple.mrj.MRJOpenDocumentHandler` interface contains one method, `handleOpenFile`, which is called when a document file needs to be opened.

handleOpenFile

Handles opening a document file.

```
public void handleOpenFile (File fileName);
```

`fileName` The name of the file to be opened.

DISCUSSION

This application-defined method is called when a document needs to be opened from the Finder. For example, this situation occurs when the user selects an Open menu item in the Finder or if the user double-clicks a file that bears the application's creator. You must register the method name by calling the `registerOpenDocumentHandler` method (page 41) when initializing the application. The `handleOpenFile` method is only useful if you have packaged your Java application using JBindery.

SEE ALSO

The `registerOpenDocumentHandler` method (page 41).

The MRJPrintDocumentHandler Interface

The `com.apple.mrj.MRJPrintDocumentHandler` interface contains one method, `handlePrintFile`, which is called when a user prints a document file from the Finder.

myHandlePrintFile

Handles printing a file.

```
public void handlePrintFile (File file);
```

`file` The name of the file to be printed.

DISCUSSION

This application-defined method is called when the application receives a request to print a file. For example, this occurs when the user attempts to print a file from the Finder that bears the application's creator. You must register the method name by calling the `registerPrintDocumentHandler` method (page 41) when initializing the application. The `handlePrintFile` method is only useful if you have packaged your Java application using JBindery.

IMPORTANT

Printing is not supported in JDK 1.0.2 and earlier. ▲

SEE ALSO

The `registerPrintDocumentHandler` method (page 41).

The MRJQuitHandler Interface

The `com.apple.mrj.MRJQuitHandler` interface contains one method, `handleQuit`, which is called when the Finder requests that the application quit.

handleQuit

Handles a quit request.

```
public void handleQuit ();
```

DISCUSSION

This application-defined method is called when the Finder requests that the application quit. For example, this occurs when the user selects the Shutdown menu item in the Finder. Typically, your method should perform any necessary cleanup (and possibly ask if the user really wants to quit) and then call the method `java.lang.System.exit()`. You must register the method name by calling the `registerQuitHandler` method (page 42) when initializing the application. The `handleQuit` method is only useful if you have packaged your Java application using JBindery.

SEE ALSO

The `registerQuitHandler` method (page 42).

Glossary

Abstract Window Toolkit (AWT) In the Java runtime environment, a collection of functions that allows Java programs to manipulate virtual graphics (windows, images, buttons, and so on). These abstract graphics can be translated into user-visible windows and controls on the client platform. See also **AWT Context.**

applet In the Java runtime environment, an executable program that must run within a larger host application. In JManager, an instantiated applet is called a `JMAppletViewerRef` object.

applet tag Text in an HTML document that describes an embedded applet. This text is bounded by the `<APPLET>` and `</APPLET>` delimiters. See also **Hypertext Markup Language (HTML).**

AWT context An instantiation of an execution environment in the Java runtime environment. An AWT context is a separate thread and may represent a thread group. An AWT context typically contains an applet and one or more frames. In JManager, an AWT context is called a `JMAWTContextRef` object. See also **Abstract Window Toolkit (AWT).**

code verifier A bytecode verifier that is part of the Java runtime environment. The code verifier acts as a security measure to make sure the Java code to be executed cannot crash the Java virtual machine or otherwise attempt illegal actions that might allow the code access to the host platform.

creator On the Mac OS platform, a 4-byte character string that identifies the application that created a file.

embedding application The application on a host platform (for example, a Web browser) that instantiates a Java session and executes Java applets or applications.

file system specification record On Mac OS–based platforms, a method of describing the name and location of a file or directory. File system specification records are defined by the `FSSpec` data type.

Finder The Mac OS application that manages the desktop. The Finder handles opening files and applications as well as maintaining the directory hierarchy.

file type On the Mac OS platform, a 4-byte character string that indicates the contents of a file. For example, files containing raw ASCII text are assigned the file type `'TEXT'`.

frame A user interface window in the Java virtual machine. Frames usually contain a title bar and often correspond to a user-visible window. Frames are analogous to a window record on the Mac OS platform. See also **parent frame.**

HTML See **Hypertext Markup Language.**

Hypertext Markup Language (HTML) A standard for describing the layout and contents of a hypertext document. An HTML document can contain an applet tag that specifies the name and location of an applet. See also **applet tag.**

Java runtime environment The Java virtual machine and the associated software required to load and execute Java code. See also **virtual machine.**

Java runtime session An instantiation of the Java runtime environment (that is, an instantiation of the Java virtual machine and associated software). In JManager a Java runtime session is called a `JMSessionRef` object. See also **virtual machine.**

keyboard equivalent A key combination that performs the same action as a menu item.

parent frame The main user interface window associated with an applet. The parent frame is created when the applet is instantiated. In an AWT context, the parent frame has the index value 0. See also **frame.**

property A data item associated with an object.

session See **Java runtime session.**

thread An independent event loop in the Java virtual machine. Multiple threads can run concurrently in a Java virtual machine. A thread is also called a *lightweight process.*

Uniform Resource Locator (URL) A text string that describes the location of an HTML document. A URL may point to a file or to a server that contains the file.

URL See **Uniform Resource Locator.**

virtual machine (VM) A software package that simulates the actions of a microprocessor. A virtual machine can mimic an existing processor (such as the 68K emulator on PowerPC-based, Mac OS–compatible computers) or parse special VM-specific code. Java code requires a virtual machine environment to execute. See also **Java runtime environment, Java runtime session.**

Index

M

R

S

T

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

DEVELOPMENTAL EDITOR
Donna S. Lee

PRODUCTION EDITOR
Glen Frank

Special thanks to Steve Zellers, Barry Langdon-Lassagne, and Rachel Rischpater.

Acknowledgments to Patrick Beard, Tony Francis, Peri Frantz, Gary Little, Tom O'Brien, and Shaan Pruden.