Wrox Programmer to Programmer™

Beginning Adobe AIR™

Building Applications for the Adobe Integrated Runtime

Rich Tretola

wrox

Updates, source code, and Wrox technical support at www.wrox.com

Beginning Adobe® AIR™

Building Applications for the Adobe Integrated Runtime

Rich Tretola

WILEY
Wiley Publishing, Inc.

Beginning Adobe AIR

Beginning Adobe® AIR™

Building Applications for the Adobe Integrated Runtime

Rich Tretola

WILEY

Wiley Publishing, Inc.

Beginning Adobe® Air™

Published by
Wiley Publishing, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Copyright © 2008 by Wiley Publishing, Inc., Indianapolis, Indiana
Published simultaneously in Canada
ISBN: 978-0-470-22904-0
Manufactured in the United States of America
10 9 8 7 6 5 4 3 2 1

Library of Congress Cataloging-in-Publication Data is available from the publisher.

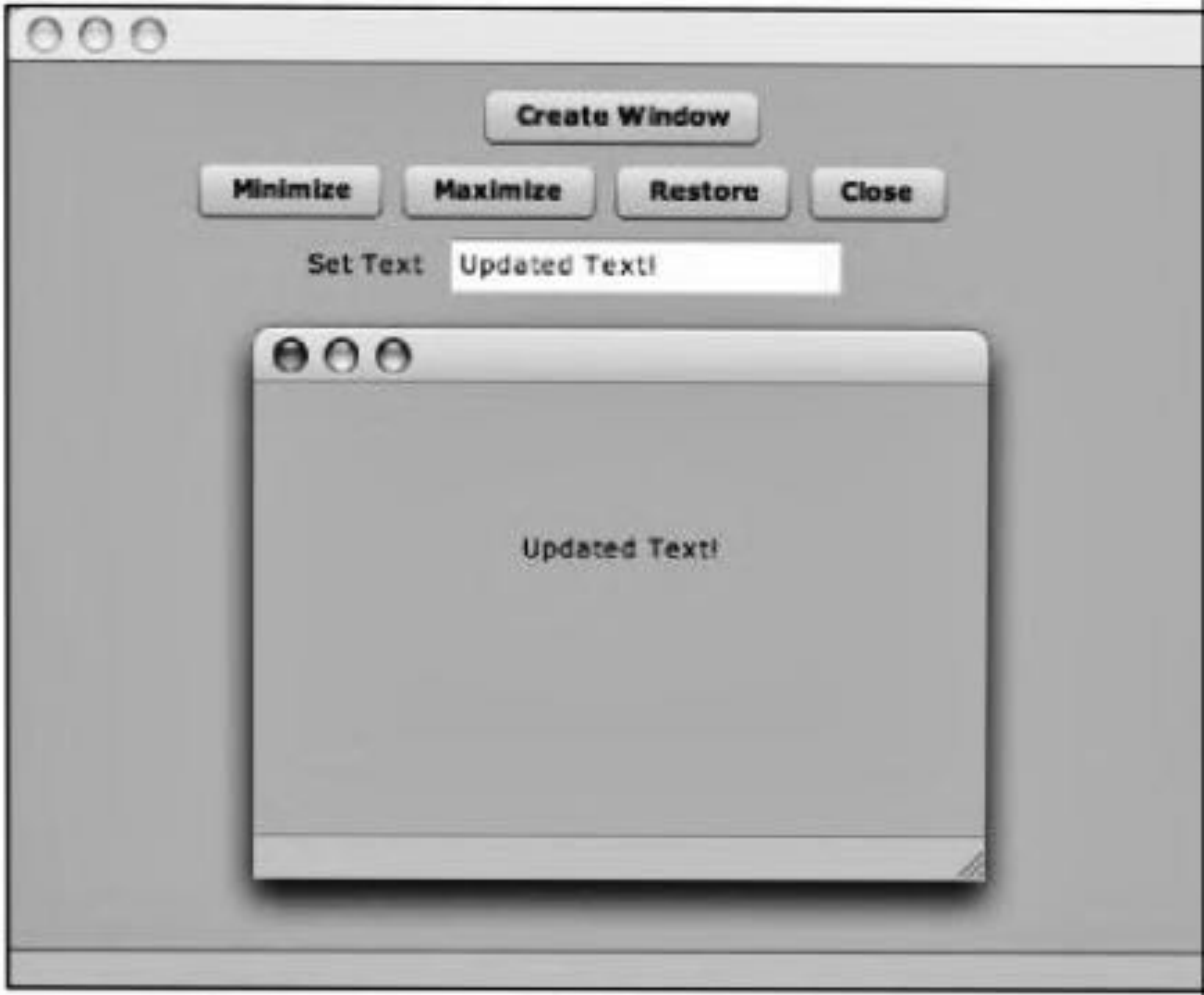
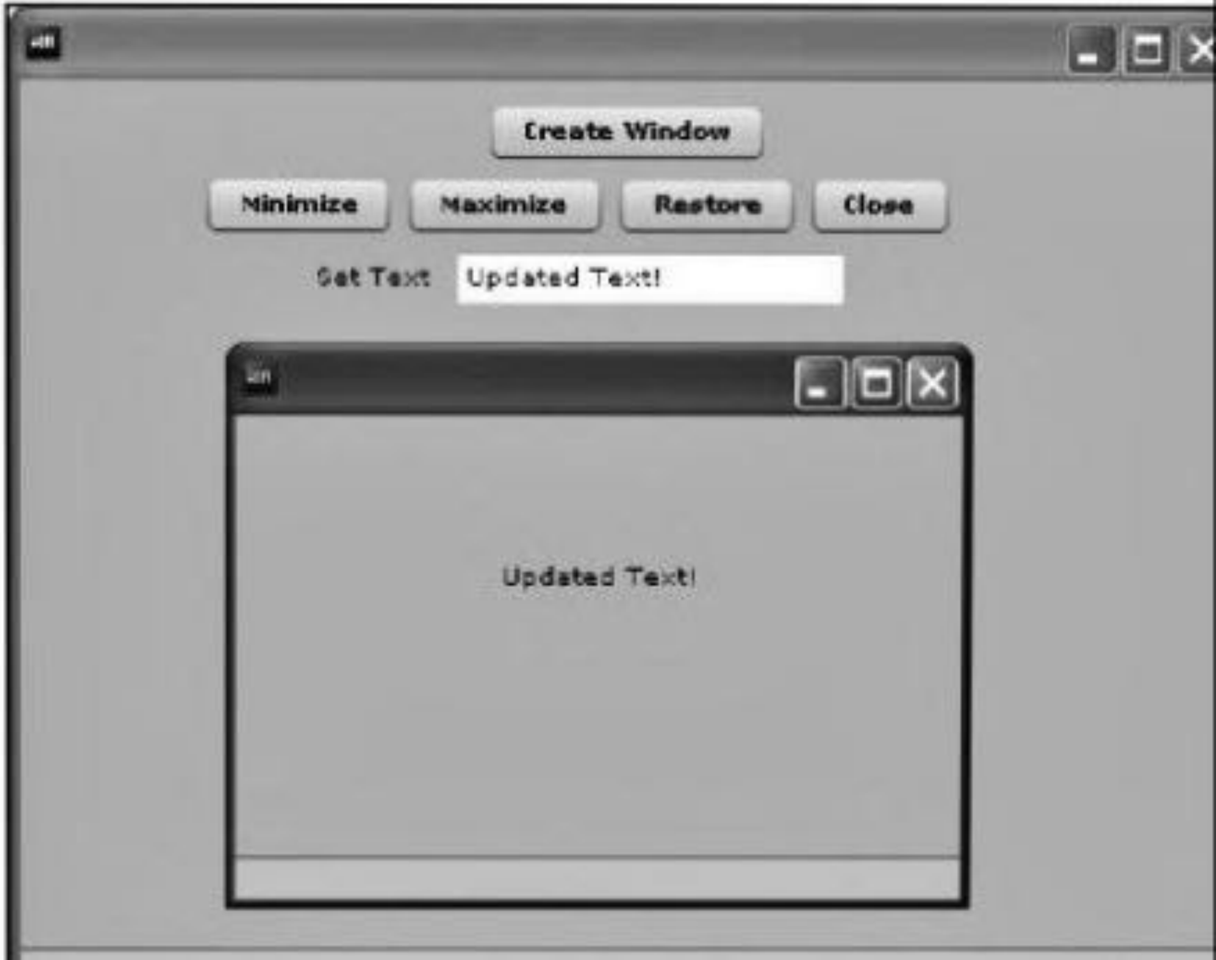


Figure 9-14: The text being updated from the TextInput on the main file on Mac OS X.



component with additional properties

```
ncoding="utf-8"?>
http://www.adobe.com/2006/mxml"

="200"
'
'


e">

Center="0" y="58"
le Window" id="lbl"/>
```

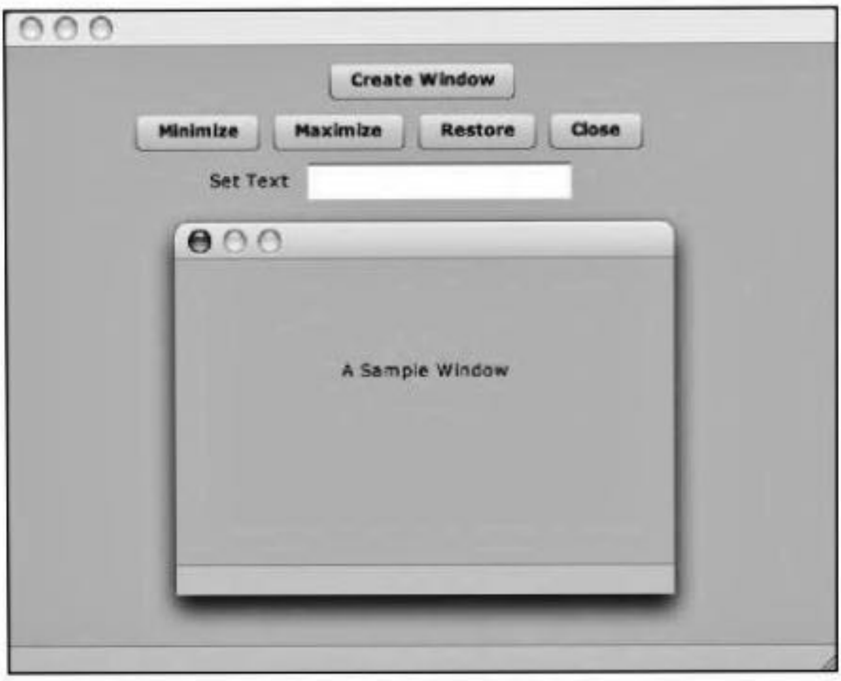
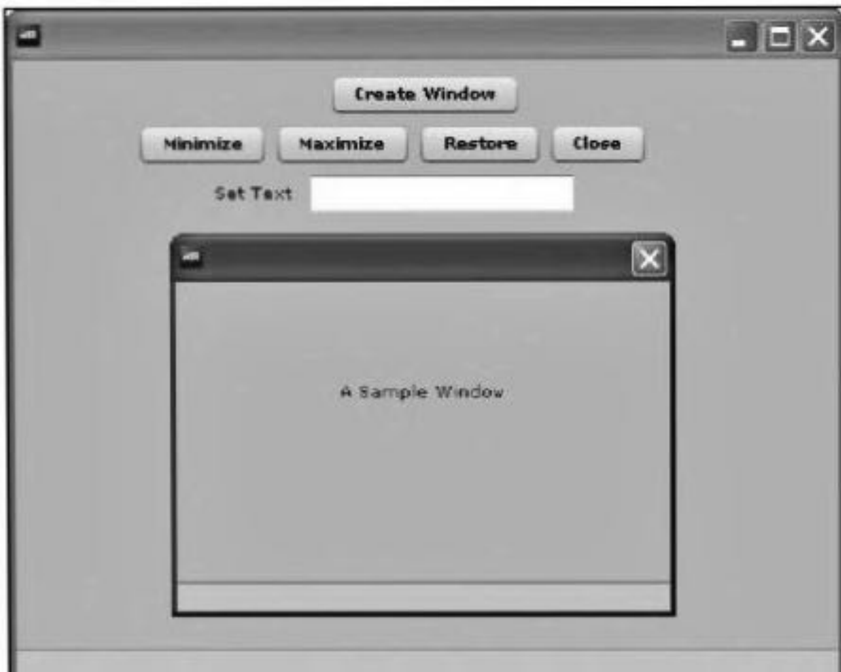


Figure 9-16: The window on Mac OS X, with onl
the Close button enabled and no Resize handle



bject previously discussed, Window also has a
he window. Setting the NativeWindowType is a
sting 9-13 shows an example of a updated mx:
The results of this updated example can be seen

of a Utility window

```
ncoding="utf-8"?>
http://www.adobe.com/2006/mxml"

t="200"


talCenter="0" y="58"
e Window" id="lbl"/>
```

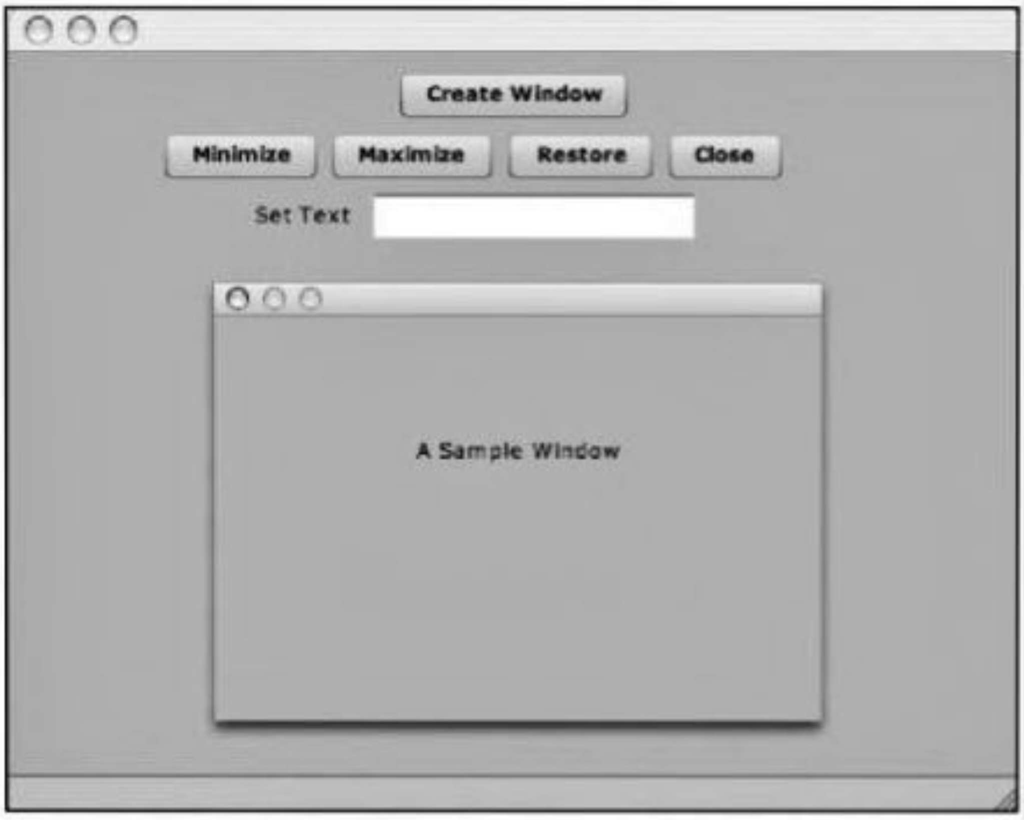
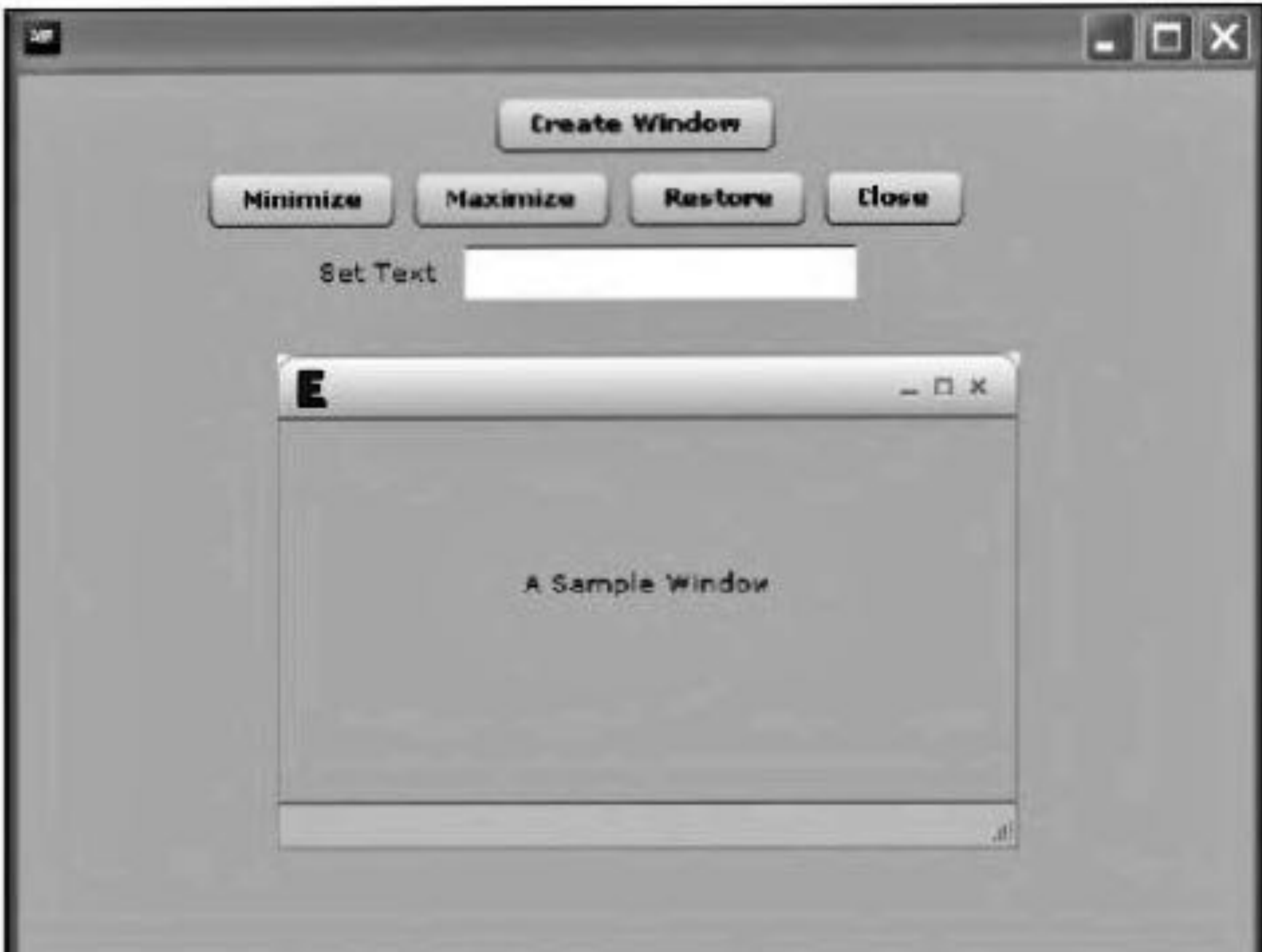


Figure 9-19: A Utility window on Mac OS X.


Create Window
Minimize
Maximize
Restore
Close
Set Text
A Sample Window

10

Interacting with the O.S.

AIR can interact with the underlying operating system by utilizing some of its existing features, including dock icons on Mac OS X, system tray icons on Microsoft Windows, and context right-click menus. The dock icons and system tray icons can also contain menus.

Dock Icons

Mac OS X has a dock that is mounted either at the right side, left side, or bottom of the screen. The dock contains shortcuts to open installed applications and also shows currently running applications. Chapter 6 demonstrated how to create custom icons for the installer. This icon would be the one shown in the dock when an application launches. AIR also gives us the ability to change the dock icon while the application is running. This can be very useful to display information to the user. Changing the icon is a simple process and simply requires the creation of a `flash.display.BitmapData` object. This can be created from an existing image displayed within the application or can be created from an embedded image class.

To get started, please create a new AIR project named *Chapter10_Dock*, which will create a new file named *Chapter10_Dock.mxml*. You will also need an image file to use as the dock icon. The source code that is part of this book has four PNG files representing four different-sized icons that I created. The image sizes are 16 × 16, 32 × 32, 48 × 48, and 128 × 128. As you will see shortly, the dock icon accepts an array of images of various sizes and then uses the best image to fit the situation; you can get away with using just one mid-sized image, which will be scaled accordingly.

We'll now proceed to the source code for adding a dock icon. Enter the code from Listing 10-1 into the newly created Chapter10_Dock.mxml file. If you only have one icon image that is 128 × 128, you could alternatively use the code from Listing 10-2, but the quality may not be acceptable when it is scaled to meet all the system requirements.

The code in Listing 10-1 first embeds four image files and types them as class objects. Next, on creationComplete of the application, the `init()` function is called where four new BitMap objects are created from the embedded images. Finally, the `NativeApplication.nativeApplication.icon.bitmaps` is set using the `bitMapData` properties from the BitMap objects.

```
            [Embed(source="e128.png")]
            [Bindable]
            private var Icon128:Class;

            private function init():void{
                var bitmap128:Bitmap = new Icon128();
                NativeApplication.nativeApplication.icon.bitmaps =
[bitmap128.bitmapData];
            }

        ]]>

    </mx:Script>
</mx:WindowedApplication>
```

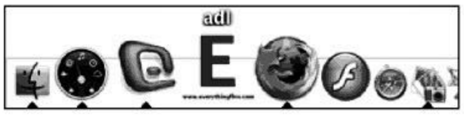


Figure 10-1: The dock icon on Mac OS X.

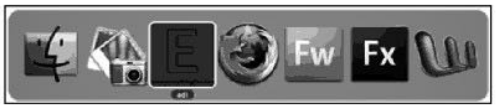


Figure 10-2: The Alt-Tab menu with the new icon.

Menu

Dock icons also support the addition of *custom menus*. Adding a custom menu is very similar to adding a `NativeMenu` or a `ContextMenu`. You will simply need to build a `NativeMenu` object and then apply it to the DockIcon object. For more information on `NativeMenus`, please take a look at Chapter 12.

To add a menu to the dock icon, add the two functions from Listing 10-3 to the Chapter10_Dock.mxml file's script block. If you examine these functions, you will see that there is a `createMenu()` function and a `handleMenuClick()` function. The `createMenu()` function first creates a `NativeMenu` object and then creates four NativeMenuItem objects. Each NativeMenuItem has an event listener added and assigned the `handleMenuClick()` function, the function that will be called when the menu is selected. The NativeMenuItem items are then added to the `NativeMenu` object, and finally, if the `NativeApplication.supportsDockIcon` returns True, the `NativeApplication.nativeApplication.icon` is cast as a DockIcon and the menu is set.

When a NativeMenuItem is selected, the `handleMenuClick()` function is called and the appropriate action is taken.

The last thing you need to do is add a call to the `createMenu()` function within the Chapter10_Dock.mxml's `init()` function. The results can be seen in Figure 10-3.