Dashboard Tutorial

2006-01-10

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

Simultaneously published in the United States and Canada.

Contents

Introduction Introduction to Dashboard Tutorial 7

Who Should Read This Document? 7
Organization of This Document 7
See Also 7

Chapter 1 Dashboard Overview 9

The Dashboard Environment 9
Dashboard Widgets 10
Widget or Application? 10

Chapter 2 Widget Basics 13

Widget Bundle Structure 13
HTML, CSS, and JavaScript Files 14
Widget Property Lists 15
Icons and Default Images 16
Widget Implementation 17
Assembling the Widget 18

Chapter 3 Next Steps 21

Dashboard Documentation 21
Useful Articles and Technical Notes 22
Dashboard Sample Code 22

Document Revision History 23

Figures and Tables

Chapter 2 Widget Basics 13

Figure 2-1 The Hello World widget bar icon 16
Figure 2-2 The Hello World widget default image 17
Figure 2-3 The Hello World widget being previewed in Safari 18
Figure 2-4 The Hello World widget installed and running in Dashboard 19
Table 2-1 File extension mappings for web technologies 14
Table 2-2 Widget `Info.plist` values 15

Introduction to Dashboard Tutorial

This document provides an overview of Dashboard and the widgets that exist in it. It introduces you to the Dashboard environment and walks you through the creation of a sample widget.

Who Should Read This Document?

Dashboard Tutorial is for anyone who wants to create a Dashboard widget. It will provide you with an understanding of the structure and basic requirements for a widget.

Organization of This Document

This document contains the following chapters:

- "Dashboard Overview" (page 9) talks about what Dashboard is and what makes a widget.
- "Widget Basics" (page 13) walks you through the creation of a sample widget. It discusses the internal structure of a widget and the files needed to make a widget work.
- "Next Steps" (page 21) includes links to various resources you'll find useful when adding features to your widget.

This document also contains a revision history.

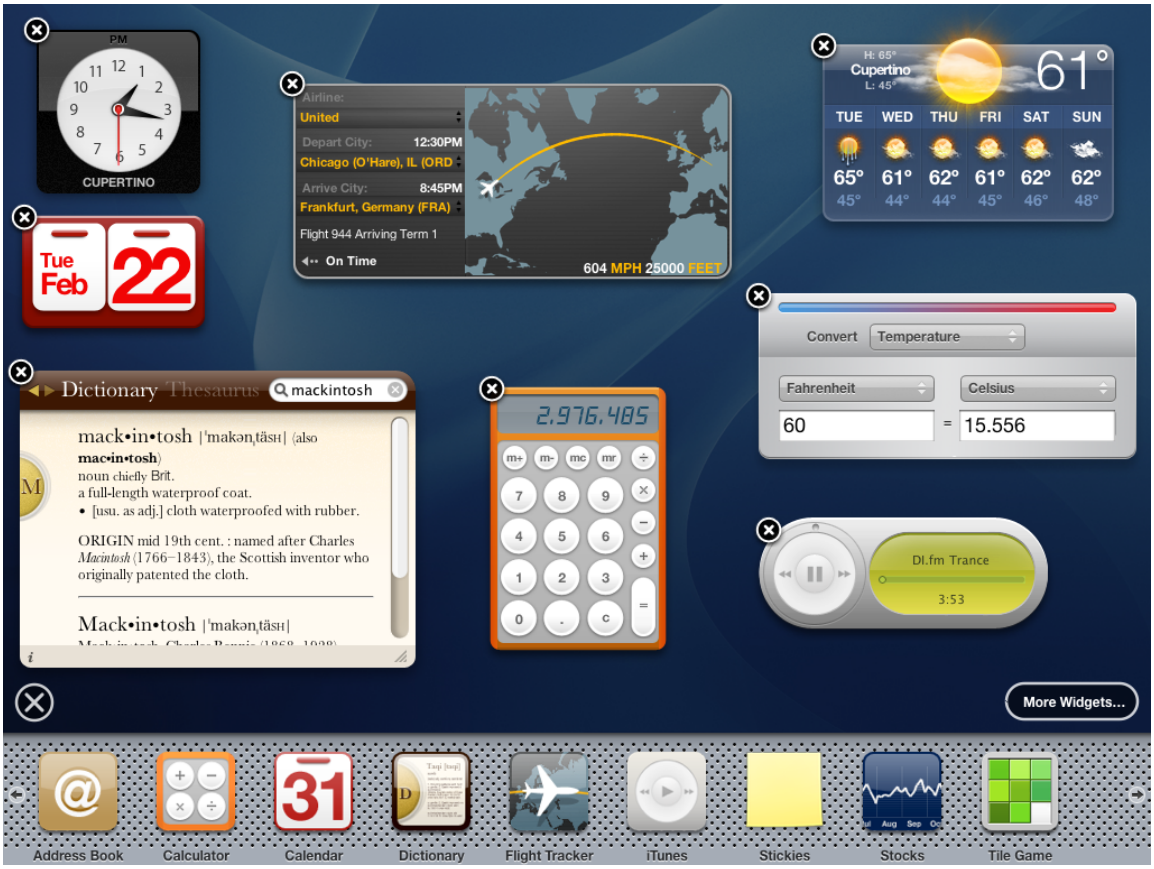
See Also

For more Dashboard and Web Kit documentation and sample code, read "Next Steps" (page 21).

Dashboard Overview

Mac OS X version 10.4 "Tiger" includes a new feature called Dashboard. Dashboard is a way to keep vital information at your fingertips, ready when you need it and easily hidden when you're done with it. That information is presented in the form of widgets–miniature applications that live exclusively in Dashboard.

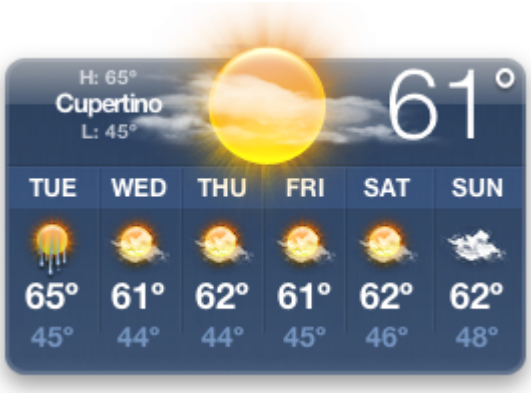
The Dashboard Environment



You show Dashboard by using a key stroke, as specified in the *Exposé & Dashboard* pane of System Preferences. By default, the key is F12. Alternatively, you can click on the Dashboard icon in the Dock. It overlays your current window set with Dashboard, the place where widgets live.

Dashboard itself is the environment where information and utilities are shown. The information and utilities are embodied in widgets. Multiple widgets can exist in Dashboard at any given time. Users have complete control over what widgets are visible and can freely move them anywhere they please within Dashboard. The widgets are shown and hidden along with Dashboard, and they share Dashboard. When Dashboard is dismissed, the widgets disappear along with it.

Dashboard Widgets



A widget is a mini-application that exists exclusively in Dashboard. From a user perspective, it behaves as a program should: it shows useful information or helps them obtain information with a minimum of required input.

Despite the fact that widgets look like applications to the user, widgets are powered by web technologies and standards such as HTML, CSS, and JavaScript. In addition to web technology, Apple provides useful additions such as preferences, localization, and system access.

Widget or Application?

Before creating a widget, you need to decide what its functionality should be. Be careful to avoid having your widget do everything a full application would do.

An example of judicious use of a widget is to provide a front end for a time card application. The application provides all of the features needed by the user, while a companion widget lets you clock in and out and choose the job that you're currently working on.

The widgets that Apple provides with Mac OS X v10.4 can give you a clue as to the scope of a widget's functionality. For instance, the Stickies widget lets you type, copy, cut, and paste, but you can't save the contents of the widget. The Weather widget shows you, by default, the temperature and location,

along with a visual indicator of the current weather condition. Upon clicking the widget, a forecast is shown. Notice that the widget shows only one location; if users want to track more locations, they can simply add another instance of the Weather widget to their Dashboard.

As a rule, avoid making a widget that your users live in, meaning that they spend considerable time working in it to get serious tasks done. Widgets should provide information with little or no input or should perform simple tasks that a user may want to do often. Also, be respectful of the limited space available on Dashboard. If your widget is needlessly large, don't expect users to keep it around.

The next chapter, "Widget Basics" (page 13), guides you through the creation of a simple widget, explains the elements that go into a widget, and where the elements belong within a widget's structure.

Widget Basics

The power of Dashboard comes in the form of its widgets. Widgets are the objects that users interact with when Dashboard is invoked. They perform tasks such as providing a clock or a calculator for the user. To develop a widget you must work with the bundle structure, a property list, and some combination of HTML, CSS, and JavaScript.

In this chapter, you create a simple "Hello, World!" widget:

In doing so, you take a look at a widget's bundle structure, its `Info.plist` property list, a basic style sheet, and the HTML file needed to make the widget function. Finally, you tie it all together and install the widget in the right place.

All of the files needed to implement this widget are installed as part of the Dashboard sample code. You can find this and other sample widgets on your hard disk at `/Developer/Examples/Dashboard/` after you install the Xcode Tools. The Xcode Tools are available for download from the Apple Developer Connection website at http://connect.apple.com/.

Note: The sample code included in this chapter is simplified and may not conform to strict HTML specifications.

Widget Bundle Structure

Widgets are distributed as bundles. A bundle is a directory in the file system that groups related resources together in one place. A widget's bundle contains at least four files: an information property list file (`Info.plist`), an HTML file, and two PNG images. In the `Hello World` example widget, there is a fifth file that contains the style sheet for the widget.

For the `Hello World` widget, the bundle structure contains the following files:

```
Hello World.wdgt/
    Icon.png
    Info.plist
    Default.png
    HelloWorld.html
    HelloWorld.css
```

HTML, CSS, and JavaScript Files

The HTML, CSS, and JavaScript files provide the implementation of the widget. In these files, you can use any technique or trick that you would use when designing a webpage. This includes, but is not limited to, HTML, CSS, and JavaScript. Use JavaScript to introduce interactivity into your widget.

While you can place all of your HTML, CSS, and JavaScript code into one file, you may find it more manageable to split these into separate files, as shown in Table 2-1 (page 14). Splitting your markup, design, and logic into separate files may also make localization easier, as discussed later on in Localizing Widgets. In general, file extensions are used to reflect the purpose of the file:

Table 2-1 File extension mappings for web technologies

Technology	Purpose	File extension	Example
HTML	Structure	`.html`	`HelloWorld.html`
CSS	Design	`.css`	`HelloWorld.css`
JavaScript	Logic	`.js`	`HelloWorld.js`

These file extensions are not enforced by Dashboard, but it is recommended that you adhere to these standards.

To load your CSS and JavaScript, you'll need to import them inside of your HTML file. For importing a style sheet, the markup looks like this:

```
<style type="text/css">
    @import "HelloWorld.css";
</style>
```

To load a JavaScript file, use the `<script>` tag:

```
<script type='text/javascript' src='HelloWorld.js'></script>
```

Note that if your widget does not use CSS or JavaScript, there is no need to use these includes or to include blank CSS or JavaScript files. Conversely, you can use multiple `@import` statements and `<script>` tags to include more than one CSS or JavaScript file.

Widget Property Lists

Each widget must have an information property list file associated with it. This file provides Dashboard with information about your widget. Dashboard uses this information to set up a space in which it can operate.

The `Info.plist` file contains the needed information. In a basic widget's information property list file are five mandatory values and four optional values. The properties are listed in Table 2-2 (page 15), along with a definition and the value used in the sample `Hello World` widget:

Table 2-2 Widget `Info.plist` values

Property	Example	Definition
`CFBundleIdentifier`	`com.apple.widget.Hello-World`	Required. A string that uniquely identifies the widget, in reverse domain format.
`CFBundleName`	`Hello World`	Required. A string that provides the name of your widget. Must match the name of the widget bundle on disk, minus the `.wdgt` file extension.
`CFBundleDisplayName`	`Hello World`	Required. A string that reflects the actual name of the widget, to be displayed in the widget bar and the Finder.
`CFBundleVersion`	`1.0`	Required. A string that gives the version number of the widget.
`CloseBoxInsetX`	`16`	Optional. An integer between 0 and 100 that sets the placement of the widget's close box on the x-axis.
`CloseBoxInsetY`	`14`	Optional. An integer between 0 and 100 that sets the placement of the widget's close box on the y-axis.
`Height`	`126`	Optional. A number that gives the height, in pixels, of your widget. If not specified, the height of Default.png is used.
`MainHTML`	`HelloWorld.html`	Required. A string that gives the name of the main HTML file that implements your widget.
`Width`	`235`	Optional. A number that gives the width, in pixels, of your widget. If not specified, the width of `Default.png` is used.

Of note are the `CloseBoxInsetX` and `CloseBoxInsetY` values. These values determine the placement of the close box over the top-left corner of your widget. You should position the close box so that the "X" is centered over the top-left corner of the widget.

The complete information property list file for the `Hello World` sample widget looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>CFBundleDisplayName</key>
    <string>Hello World</string>
    <key>CFBundleIdentifier</key>
    <string>com.apple.widget.helloworld</string>
    <key>CFBundleName</key>
    <string>Hello World</string>
    <key>CFBundleShortVersionString</key>
    <string>1.0</string>
    <key>CFBundleVersion</key>
    <string>1.0</string>
    <key>CloseBoxInsetX</key>
    <integer>16</integer>
    <key>CloseBoxInsetY</key>
    <integer>14</integer>
    <key>MainHTML</key>
    <string>HelloWorld.html</string>
</dict>
</plist>
```

Note that in this `Info.plist`, the `Width` and `Height` keys are omitted. As previously mentioned, these keys are optional. Since they aren't included, the widget is automatically sized based on the dimensions of its default image.

There are more, optional Info.plist keys than those used here; read about them in *Dashboard Reference*. Of particular note are the Access Keys, which allow you to turn on access to external resources. Specifying Access Keys discusses these more in-depth.

Note: While an `Info.plist` file is just a text file, it's easiest to edit one using the Property List Editor application. It's found in `/Developer/Applications/Utilities/` on your hard disk after you installed the Xcode Tools.

Icons and Default Images

The two image files required in a widget are the icon and default image files. They need to be formatted as Portable Network Graphics (PNG) files and must be named `Icon.png` and `Default.png`, respectively.

The icon file, `Icon.png`, is used in the widget bar as a representation of your widget:

Figure 2-1 The Hello World widget bar icon

The default image, `Default.png`, is shown while your widget loads. It can be the background used by your widget or any other appropriate image. This file also sets the size of your widget if you don't use the `Height` and `Width` properties in your `Info.plist` file.

Figure 2-2 The Hello World widget default image

More on widget bar icon sizes and other design elements is located in Designing Widgets.

Note: When assembling your copy of the `Hello World` sample widget, make sure that the `Icon.png` and `Default.png` file names have their leading letter capitalized.

Widget Implementation

Your widget's HTML file provides the implementation of the widget. It can be named anything but must reside at the root level of the widget bundle and must be specified in the `Info.plist` file. For the `Hello World` sample widget, the HTML file displays an image and the words "Hello, World!" Here are the contents of the `HelloWorld.html` file:

```
<html>
<head>
<style>
    @import "HelloWorld.css";
</style>
</head>

<body>

    <img src="Default.png" />
    <span class="helloText">Hello, World!</span>

</body>
</html>
```

The HTML for this widget specifies the image used as the background and the text to display. You'll notice however that the above HTML file doesn't contain any style information. Instead, it imports another file that has this information: `HelloWorld.css`. As discussed in "HTML, CSS, and JavaScript Files" (page 14), it isn't required that you break your CSS and JavaScript out of the HTML file, but it is recommended. The file `HelloWorld.css` contains all of the style information for the widget:

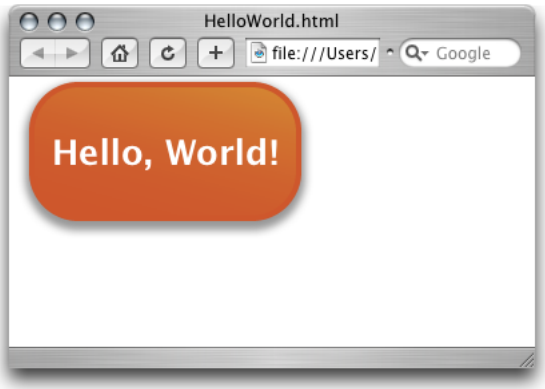
```
body {
    margin: 0;
}

.helloText {
```

```
    font: 26px "Lucida Grande";
    font-weight: bold;
    color: white;
    position: absolute;
    top: 41px;
    left: 32px;
}
```

The style sheet defines the styles for the body and for an arbitrary span class called `helloText`. This class is applied to the "Hello, World!" text in the `Hello World` HTML file.

Figure 2-3 (page 18) shows what the code looks like when rendered in Safari.

Figure 2-3 The Hello World widget being previewed in Safari



In fact, when testing your widget, you can load it into Safari and get the same behavior that you would if it were loaded into Dashboard. The exception to this rule are interactions that rely specifically on the presence of Dashboard, as discussed in coming chapters.

Assembling the Widget

Now that you have the three basic components for the widget ready, you can assemble them into a bundle and load your completed widget into Dashboard.

First, create a new directory named `Hello World`. Then, place these files in it at the root level:

- `Default.png`
- `HelloWorld.html`
- `HelloWorld.css`
- `Icon.png`
- `Info.plist`

Once the files are in place, rename the directory `Hello World.wdgt`.

Note: If you do not have the "Show all file extensions" preference checked in the Finder, you are asked to confirm this action. Go ahead and choose Add and the bundle will be made for you. If you have this preference checked, the bundle is made without any prompting.

Once the bundle has been renamed, double click its icon in the Finder to install it. This displays an install dialog that, when you click Install, copies the widget to `~/Library/Widgets/` and opens it in Dashboard. It should look like the view in Figure 2-4 (page 19).

Figure 2-4 The Hello World widget installed and running in Dashboard

Congratulations! You've just created your first Dashboard widget.

Now you're ready to enhance your widget with some of the features Apple provides in Dashboard and Web Kit. Read "Next Steps" (page 21) for information on where to go next.

Next Steps

Now that you've created a basic widget, you have a solid foundation to base your own widget on. Here are some useful resources for you to consult when adding features to your widget.

Dashboard Documentation

Apple provides other Dashboard-focused documents that expose all of the features that you can incorporate into a widget.

Dashboard Programming Topics explains all of the Dashboard-specific features available, in addition to standard Web Kit features that you may find useful. A few articles of note are:

Designing Widgets
: Before adding any more code to your widget, you should think about its design and functionality. This article provides you with tips and tricks for designing a high-quality widget that conforms to users expectations.

Introduction to the Apple Classes
: Apple provides you with common controls and utilities, like buttons, sliders, and animation timers. These JavaScript classes are called Apple Classes and are discussed in this article and the articles linked from it.

Widget Backs and Preferences
: Widgets use their backs to display preferences and other information to users. This article covers how you need to setup your widget's HTML and CSS to support sides and the JavaScript code you need to implement the flip animation.

Localizing Widgets
: Localizing your widget increases the potential audience for your widget. This article discusses the localization mechanism within Dashboard and includes tips for localizing text.

Accessing Command Line Utilities
: Dashboard provides you with an easy way to use a command line utility or script within a widget. This article walks you through using the utility in synchronous or asynchronous mode and interacting with the utility as it executes.

Creating a Widget Plug-in
: Widget Plug-ins provide a bridge between JavaScript and Cocoa, letting you interact with native code from within your widget. This article discusses the Widget Plug-in protocol and

touches on the Webscripting protocol, used for the actual exchange of data between JavaScript and Cocoa.

Delivering Widgets
: Once you've created, tested, and debugged your widget, it's time to share it with the world! This article provides information on properly packaging you widget so that it takes advantage of Mac OS X v.10.4's easy widget installation experience.

This is not an exhaustive list of the articles available in *Dashboard Programming Topics*. Please browse Introduction to Dashboard Programming Topics to see all the document has to offer you.

In addition to the Dashboard conceptual documentation, Apple offers *Dashboard Reference*. This document provides in-depth details on all of the interfaces and utilities made available within widgets.

The Safari Reference Library (Reference Library > Apple Applications > Safari) provides information on the content display capabilities of Web Kit, the technology that powers Dashboard widgets. The HTML, CSS, and JavaScript documentation found there applies to widgets as well.

Useful Articles and Technical Notes

In addition to the Dashboard documentation, these articles may prove useful when creating your widget:

Debugging Dashboard Widgets
: This Technical Note provides useful tips for resolving issues within your widget.

Dynamic HTML and XML: The XMLHttpRequest Object
: This article discusses the `XMLHttpRequest` object, a JavaScript object used to obtain information from the Internet.

Dashboard Sample Code

Apple publishes Sample Code for Dashboard. These samples demonstrate how to use various Dashboard and Web Kit technologies within the context of a widget. You can find the Dashboard sample code at:

`/Developer/Examples/Dashboard/`
: Available on disk after you install the Xcode Tools, the Dashboard examples available at this path provide the most complete set of samples available. To get the latest examples, install the Xcode Tools 2.2, available for download from http://connect.apple.com/.

Sample Code > Apple Applications > Dashboard
: The ADC Reference Library houses additional sample code, including the most up-to-date versions of sample widgets included with Xcode.

Document Revision History

This table describes the changes to *Dashboard Tutorial*.

Date	Notes
2006-01-10	Updated the widget tutorial to reflect the installation experience on newer versions of Mac OS X v10.4.
2005-12-06	Retitled document Dashboard Tutorial. Relocated chapters on advanced Dashboard topics to Dashboard Programming Topics.
2005-08-11	Added Information about the Widget Installer. Fixed various typos throughout the document.
2005-07-07	Added sections about using a search field, creating a generic button, Universal Access, and adding help tags. Renamed Security chapter to Access Keys. Widget reverses now referred to as Widget backs.
2005-06-04	Added section on implementing integrated menus. Fixed various typos.
2005-05-20	Revised Basics chapter to include Info.plist example, added info on widget bar icons, clarified the Preferences chapter, modified Security chapter, and fixed various typos.
2005-04-29	Includes information about distributing widgets and more widget design guidelines.
	Updated for public release of Mac OS X v10.4. First public version.
	Reorganized the document into task-based chapters. Includes new chapters on the widget security model, command-line access, design guidelines, and overview. Also added sections on drag events and the widget close box. Added more sample code.
2004-11-02	Revised Widget operations and widget plug-in. Relocated Canvas, Pasteboard, and Drag and Drop chapters to Safari JavaScript Programming Topics and included links.
2004-10-04	Added Localization and Application Activation topics. Revised Canvas, Control Regions, Widget Resizing, Preferences, and more topics.

Date	Notes
2004-08-31	Fixed code and formatting issues.
2004-06-28	New document that provides an overview of the Dashboard environment and the widgets that exist in it.