
Dashcode User Guide

[Tools > Scripting & Automation](#)



2009-03-04



Apple Inc.
© 2009 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

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

Apple, the Apple logo, Dashcode, iCal, iPhoto, iPod, Mac, Mac OS, Monaco, Objective-C, Quartz, QuickTime, Safari, and Xcode are trademarks of Apple Inc., registered in the United States and other countries.

Finder, iPhone, and MobileMe are trademarks of Apple Inc.

Adobe, Acrobat, and PostScript are trademarks or registered trademarks of Adobe Systems Incorporated in the U.S. and/or other countries.

Helvetica and Times are registered trademarks of Heidelberger Druckmaschinen AG, available from Linotype Library GmbH.

Java and all Java-based trademarks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Introduction **Introduction to Dashcode User Guide 9**

- Who Should Read This Document? 9
- Organization of This Document 9
- Getting and Running Dashcode 10
- Reporting Bugs 10
- See Also 10

Chapter 1 **Widget Tutorial 13**

- Before You Begin 13
- Choose a Template 13
- Set the Target Date 15
- Test the Countdown 15
- Customize the Widget's Appearance 15
- Add Functionality Using Parts 16
- Write Code to Show the Apple Store 17
- Deploy Your Widget 18

Chapter 2 **Web Application Tutorial 19**

- Before You Begin 19
- Choose a Template 19
- Learn About the Template 21
- Test the Application 21
- Explore the Views in the Stack Layout 22
- Add Functionality Using Parts 23
- Write Code to Perform a Search 24
- Deploy Your Web Application 25

Chapter 3 **Starting a Project 27**

- Creating a Project from a Template 27
- Creating a Project from an Existing Widget 27
- Providing Widget Attributes 28
- Providing Attributes for a Web Application 28
- Opening an Existing Widget 30

Chapter 4 **Designing a Widget or a Web Application 31**

- Laying Out the Interface 31
- Showing a Widget's Sides 31

- Adding Parts to an Interface 32
- Using Photos from iPhoto 32
- Changing an Element's Properties 32
- Arranging and Locking Elements 33
- Absolute versus Document-Flow Positioning 33
- Searching For Elements 34
- Rulers and Guides 34
- Placing the Close Box 34
- Disabling the Canvas 35
- Previewing a Widget's Default Image 35
- Designing a Widget Icon 35
- Designing a Web Clip Icon for a Web Application 36

Chapter 5 Adding Source Code 39

- Viewing a Project's Source Code 39
- Using HTML, CSS, and JavaScript Programming Interfaces 40
- Adding a Handler for an Event 40
- Adding Code for Custom Controllers 40
- Code Completion Using Code Sense 41
- Code Snippets 41
- Adding and Moving Files and Folders 41
- Resource Access 42
- Code Editing Preferences 43

Chapter 6 Testing and Sharing 45

- Testing a Widget or a Web Application 45
- The Run Log and Tracing Execution 46
- Pausing and Step-by-Step Execution 46
- Checking Values in Memory 47
- Breakpoints 47
- The Code Evaluator 48
- Sharing a Widget 48
- Deploying a Web Application 48

Chapter 7 Advanced Topics for Widgets 51

- Localization 51
- Widget Plug-ins 51

Appendix A Dashcode Templates 53

- Widget Templates 53
 - The Custom Widget Template 53
 - The Countdown Template 53

- The Maps Template 53
- The RSS Widget Template 54
- The Podcast Template 54
- The Photocast Template 55
- The Quartz Composer Template 55
- The Video Podcast Template 55
- The Gauge Template 56
- The Daily Feed Template 56
- Web Application Templates 56
 - The Custom Web Application Template 56
 - The Browser Template 56
 - The Podcast Web Application Template 57
 - The RSS Web Application Template 57
 - The Utility Web Application Template 57

Appendix B Dashcode Parts 59

- Activity Indicator 59
- Back Button 59
- Browser 59
- Canvas 60
- Call Button 60
- Column Layout 60
- Edge-to-Edge List 60
- Forward Button 61
- Gauge 61
- Indicator 62
- Level Indicators 62
- Mail Button 62
- Map Button 62
- Quartz Composer 63
- QuickTime 63
- Rounded-Rectangle List 63
- Scroll Area 64
- Stack Layout 64
 - Stack Layout Methods 64
 - Stack Layout Transitions 65

Document Revision History 67

Figures

Chapter 1 **Widget Tutorial 13**

- Figure 1-1 The Birthday widget 13
- Figure 1-2 A project window showing a new widget 14
- Figure 1-3 The Countdown template's properties 15
- Figure 1-4 Tweaking the front image using the Fill & Stroke inspector 16
- Figure 1-5 A function in the source code editor 17

Chapter 2 **Web Application Tutorial 19**

- Figure 2-1 A new web application project based on the Browser template 20
- Figure 2-2 The default Browser-based web application contains a top-level page (left) and a detail page (right) 22
- Figure 2-3 Adding a part to the web application's user interface 24
- Figure 2-4 Adding code to handle the button's click event 25

Introduction to Dashcode User Guide

This document provides an overview of the Dashcode development environment. It describes how to use Dashcode to create two types of projects:

- Dashboard widgets—simple, lightweight applications that perform a single task in the Dashboard environment. Widgets are actually packaged webpages powered by standard web technologies such as Hypertext Markup Language (HTML), Cascading Style Sheets (CSS), and JavaScript.
- Web applications—webpages optimized for Safari on iPhone that provide discrete functionality to users. Web applications also make use of web technologies such as HTML, CSS, and JavaScript.

Dashcode’s integrated environment allows you to lay out, code, and even test widgets and web applications without opening any other applications. Its layout tools, composers, and editors simplify the process of creating all the resources these projects need. Dashcode also includes handy coding and debugging tools that help you manage and test the code you write.

Who Should Read This Document?

Read *Dashcode User Guide* to learn how to use Dashcode to create web applications and Dashboard widgets. Developers who are new to either widget or web application creation learn how to build simple projects and find out more about Dashcode’s capabilities. Experienced developers learn how to speed up development using Dashcode.

Organization of This Document

This document contains the following chapters:

- [“Widget Tutorial”](#) (page 13) walks you through creating your first Dashboard widget with Dashcode.
- [“Web Application Tutorial”](#) (page 19) shows you how to create a simple web application with Dashcode.
- [“Starting a Project”](#) (page 27) discusses the different starting points when working with Dashcode.
- [“Designing a Widget or a Web Application”](#) (page 31) shows you the tools Dashcode provides for designing the user interface of a widget or web application.
- [“Adding Source Code”](#) (page 39) details the source code editing tools included with Dashcode.
- [“Testing and Sharing”](#) (page 45) includes information on testing, debugging, and distributing a widget or web application.
- [“Advanced Topics”](#) (page 51) talks about localizing a widget using Dashcode and including a widget plug-in.

Dashcode User Guide also includes these appendixes:

- “[Dashcode Templates](#)” (page 53) describes the project templates included with Dashcode.
- “[Dashcode Parts](#)” (page 59) includes information on manipulating custom Dashcode-originated elements, called parts, via JavaScript.

Getting and Running Dashcode

Apple provides a comprehensive suite of developer tools (including Dashcode) for creating Mac OS X software. The Xcode Tools include applications to help you design, create, debug, and optimize your software. This tools suite also includes header files, sample code, and documentation for Apple technologies. You can download the Xcode Tools from the Apple Developer Connection (ADC) website (<http://developer.apple.com>). Registration is required, but free.

After you download the Xcode Tools, use the Xcode Tools installer to install Dashcode. After installation, you'll find Dashcode in `/Developer/Applications/`.

Reporting Bugs

If you encounter bugs in Apple software or documentation, you are encouraged to report them to Apple. You can also file enhancement requests to describe features you would like to see in future revisions of a product or document. To file bugs or enhancement requests, go to the Bug Reporting page of the ADC website, which is at the following URL:

<http://developer.apple.com/bugreporter/>

You must have a valid ADC login name and password to file bugs. You can obtain a login name for free by following the instructions found on the Bug Reporting page. To file a bug for Dashcode, use the Dashcode component, version X.

See Also

For in-depth information on how to create web applications that work well on iPhone and iPod touch, see *Safari Web Content Guide for iPhone OS*. For guidance on how to design the user interface of such an application, see *iPhone Human Interface Guidelines*.

Read *Dashboard Programming Topics* for information on the technologies available to you when creating a widget. All of the Dashboard-specific information discussed in this document is covered in more depth in *Dashboard Reference*. Additional Dashboard documents and sample code can be found on the [ADC topic page for Dashcode](#).

The [ADC topic page for Safari](#) contains useful information on WebKit, the technology that powers Dashboard widgets. For more information on the HTML, CSS, and JavaScript capabilities found in WebKit, consult:

- *Safari HTML Reference*

INTRODUCTION

Introduction to Dashcode User Guide

- *Safari CSS Reference*
- *Safari DOM Extensions Reference*

The `XMLHttpRequest` object allows you to parse XML in JavaScript and use the results. Read the ADC article [Dynamic HTML and XML: The XMLHttpRequest Object](#) for more information.

INTRODUCTION

Introduction to Dashcode User Guide

Widget Tutorial

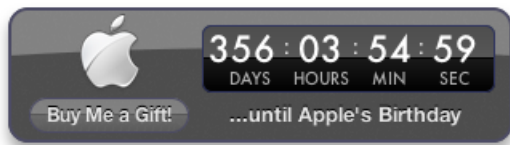
This tutorial walks you through using Dashcode to create a Dashboard widget. As you follow the steps, you learn how to choose a widget template, customize your widget's appearance and code, and share your widget with others. Completing this tutorial is a quick and easy way to get started building widgets in Dashcode.

This document includes another tutorial, “[Web Application Tutorial](#)” (page 19), which follows this one. The remainder of the document delves more deeply into the Dashcode development environment, describing how it supports both widget and web application development. If you don't want to learn how to create a web application, you can skip “[Web Application Tutorial](#)” (page 19) and continue learning more about Dashcode by reading “[Creating a Project from a Template](#)” (page 27).

Before You Begin

In this tutorial, you build a widget that counts down to your birthday, similar to the widget shown in Figure 1-1.

Figure 1-1 The Birthday widget



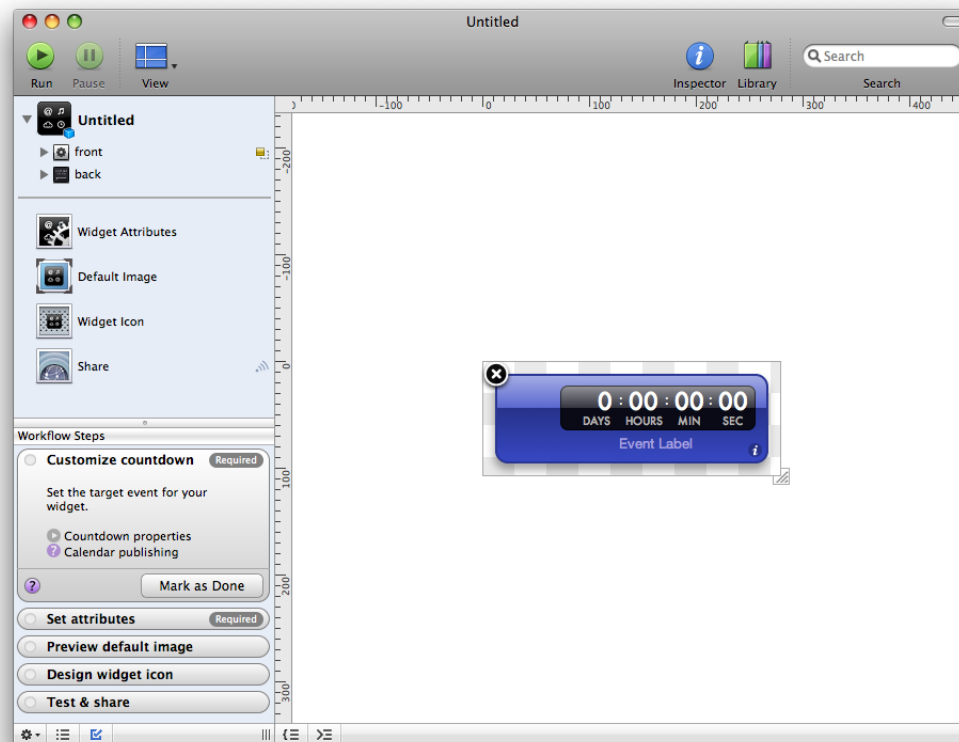
Before continuing, make sure that you have Dashcode installed on your Mac (the location is `/Developer/Applications/`). If you don't have Dashcode installed, read “[Getting and Running Dashcode](#)” (page 10) to learn how to get and install Dashcode.

Choose a Template

To start, double-click the Dashcode icon to open it. A new project window opens and displays a dialog in which you first select the type of project you're interested in—in this case, Dashboard Widget—and then, the kind of widget you want to create from an assortment of templates. **Templates** are handy starting points for creating common types of widgets. Select a template's icon to show a short description of what that template does.

To make the Birthday widget, this tutorial uses the Countdown template. Select its icon and click Choose. A project window opens with a new widget based on the Countdown template, as shown in Figure 1-2.

Figure 1-2 A project window showing a new widget



Along the left side of the project window is the **navigator**, which you use to switch between the various tools available when you're designing a widget. The main portion of the window is the **canvas**, which you use to design your widget's interface.

At the bottom of the navigator in Figure 1-2 you can see the **Workflow Steps list**, which guides you through the widget development process. Each step is a milestone in creating a widget, telling you what to do and where to do it. When you complete a step, mark it as done and move on to the next step.

Note: If you don't want to see the Workflow Steps list, you can hide it by choosing View > Steps or by clicking the button that looks like a checkbox at the bottom edge of the project window (this button is highlighted in Figure 1-2).

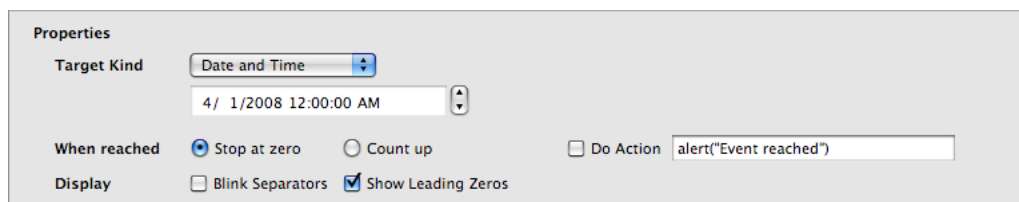
Alternatively, you can view a list of the widget's files in place of the Workflow Steps list. If you want to see the Files list, choose View > Files or click the list button in the bottom edge of the project window (it's the button to the left of the checkbox button).

Set the Target Date

The Countdown template gives you a widget with all the elements and code needed to count down to an event. All you need to do is tell the widget the target date. To set the target date, select Widget Attributes in the navigator. The canvas is replaced by the **widget attributes pane**, in which you specify important values that your widget needs.

In the Properties section of the widget attributes pane, choose Date and Time in the Target Kind pop-up menu and enter the date of your next birthday, as shown in Figure 1-3.

Figure 1-3 The Countdown template's properties



Test the Countdown

Your new widget is already fully functional. To prove this, choose Debug > Run to run the widget. Dashcode can run a widget without opening it in Dashboard, making it a handy place to test your widget and fix any problems you encounter. After the widget loads, it starts counting down towards your next birthday.

When you're satisfied that your widget is working as you expect, choose Debug > Stop to stop it.

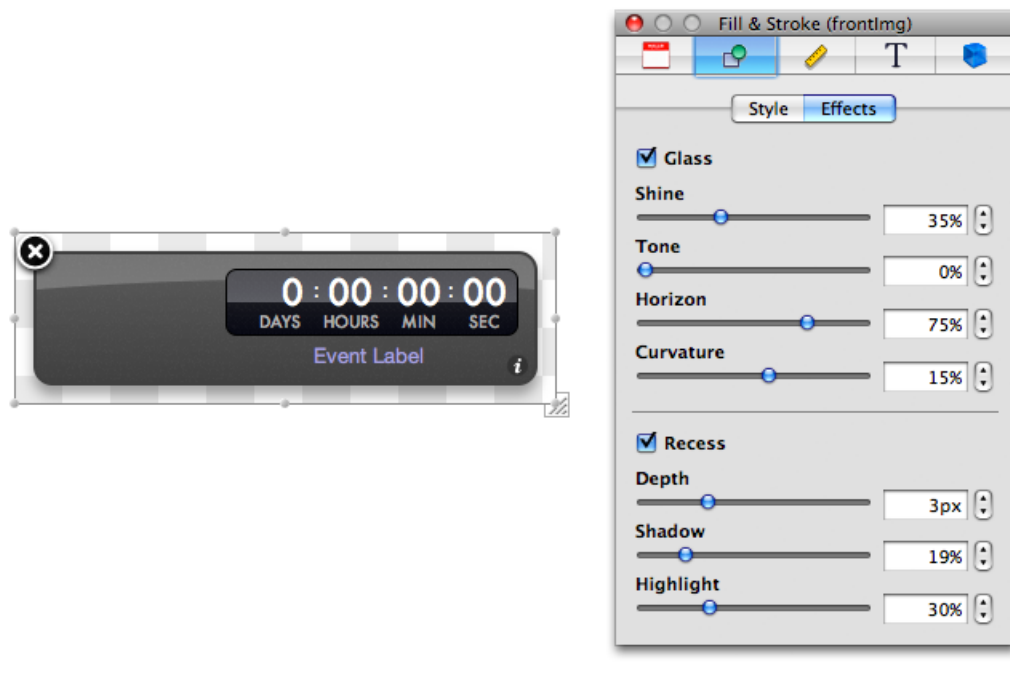
Now is a good time to save your widget project. Choose File > Save to save the project. Give your project a name and select a location to save it in. Your widget is saved in a widget project that encapsulates the widget and information Dashcode needs to create the widget for you.

Customize the Widget's Appearance

Although you now have a fully functioning widget that's ready to share, you might want to personalize it to make it unique. Dashcode's design tools make it easy to customize your widget's interface. Select the widget item in the navigator (it should display the name you gave it when you saved the project). The widget attributes pane is replaced with the canvas.

First, change your widget's body color. Select the widget body (also called the front image or `frontImg`) on the canvas and then choose `Window > Show Inspector`. The **inspector window** allows you to modify a selected element's properties, such as its appearance and behavior. Click the **Fill & Stroke** button at the top of the inspector window (it's the second from the left). If it's not already selected, click the **Style** tab to reveal fill, stroke, corner roundness, and opacity values. Click the color well and choose a new color in the **Colors** window that appears. Try different fill styles until you find a combination that you like. If you want to try changing other effects, such as the glass appearance, click the **Effects** tab to reveal these values, as shown in Figure 1-4.

Figure 1-4 Tweaking the front image using the Fill & Stroke inspector



When you're finished customizing your widget's body, add a photo of yourself from iPhoto to the widget. Your iPhoto library is available from the **Library window**. To show your iPhoto library, choose `Window > Show Library`; then click the **Photos** button. Find a photo and drag it to your widget on the canvas. Resize it by dragging any of the resize handles on the photo.

Finally, change the **Event Label** text to something like "...days until my birthday." You can do this by selecting the **Event Label** text, clicking the **Attributes** button in the inspector window (it's the leftmost button), and entering the text in the **Value** field, or by double-clicking the text in the widget body itself and entering the new text.

Add Functionality Using Parts

Now that you have a personalized widget that counts down to your birthday, add a button that, when clicked, shows the Apple Store (so your friends and family can buy you a birthday present!). To add a button to your widget, use a **button part**. Parts are controls and views used on a widget's interface.

To find a button part, choose Window > Show Library and click Parts. You can use the search field at the bottom of the window to help you find a particular part or type of part. From the list of parts, drag the Lozenge Button part from the Library window to your widget's body. Double-click the button to select its label text, enter the text "Buy me a gift" and press Return. You'll probably need to resize the button to fit the new label.

Write Code to Show the Apple Store

To make the button take the user to the Apple Store when it's clicked, you need to add a behavior to the button. In the inspector window, click the Behaviors button (it's the rightmost button). This shows the Behaviors pane, in which you assign handler functions to events on an object. Select the button on the canvas and double-click in the Handlers column next to the `onclick` event name. Enter the name of a new function, such as `showAppleStore`, and press Return. Click the arrow next to the function name you entered to reveal the **source code editor** below the canvas. Here you write code to add functionality to your widget. Clicking the arrow reveals the `showAppleStore` function Dashcode inserted in your widget's code. Between the braces (`{ . . }`) enter the following line of code:

```
widget.openURL("http://store.apple.com/");
```

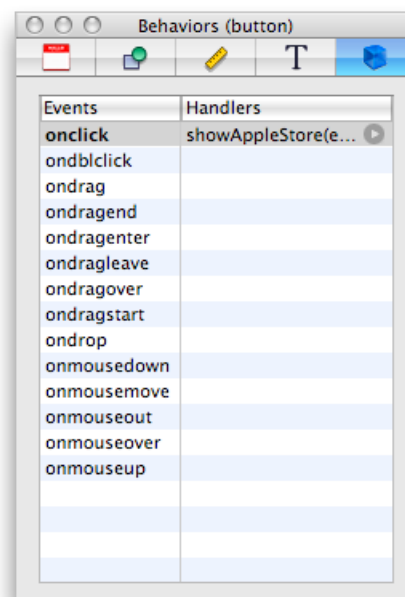
The code in the source code editor should look like that in Figure 1-5.

Figure 1-5 A function in the source code editor

```

416
417 function showAppleStore(event)
418 {
419     widget.openURL("http://www.apple.com/store");
420 }
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442

```



Test your widget again by choosing Debug > Run. Click the button you added and make sure a new Safari window opens with the Apple Store website displayed. Be sure you save your project often to preserve the changes you make.

Deploy Your Widget

Congratulations! You've created your first complete widget using Dashcode.

To open your widget in Dashboard, choose File > Deploy Widget. Click Install in the dialog that appears to view your widget in Dashboard.

To share your widget with the world, select Share in the navigator. The share pane that appears displays the widget project name you chose in ["Test the Countdown"](#) (page 15), but you can replace this with a different name if you want. You can also set the minimum version of Mac OS X your widget should run in. Click Save to Disk to save your widget. You can now email it to friends or post it on your webpage. You can use the File > Compress command in the Finder to archive it.

Web Application Tutorial

This tutorial walks you through using Dashcode to create a web application. As you follow the steps, you learn how to choose a template, customize your application's code, and share your application with others. Completing this tutorial is a quick and easy way to get started building web applications in Dashcode.

This is the second of two tutorials in this document. If you're interested in learning how to start developing widgets be sure to read "[Widget Tutorial](#)" (page 13). The remainder of the document, beginning with "[Starting a Project](#)" (page 27), delves more deeply into the Dashcode development environment, describing how it supports both widget and web application development.

Before You Begin

In this tutorial, you build a web application that displays information about national parks. This web application is based on the browser type of web application, and supports navigation through multiple levels of content.

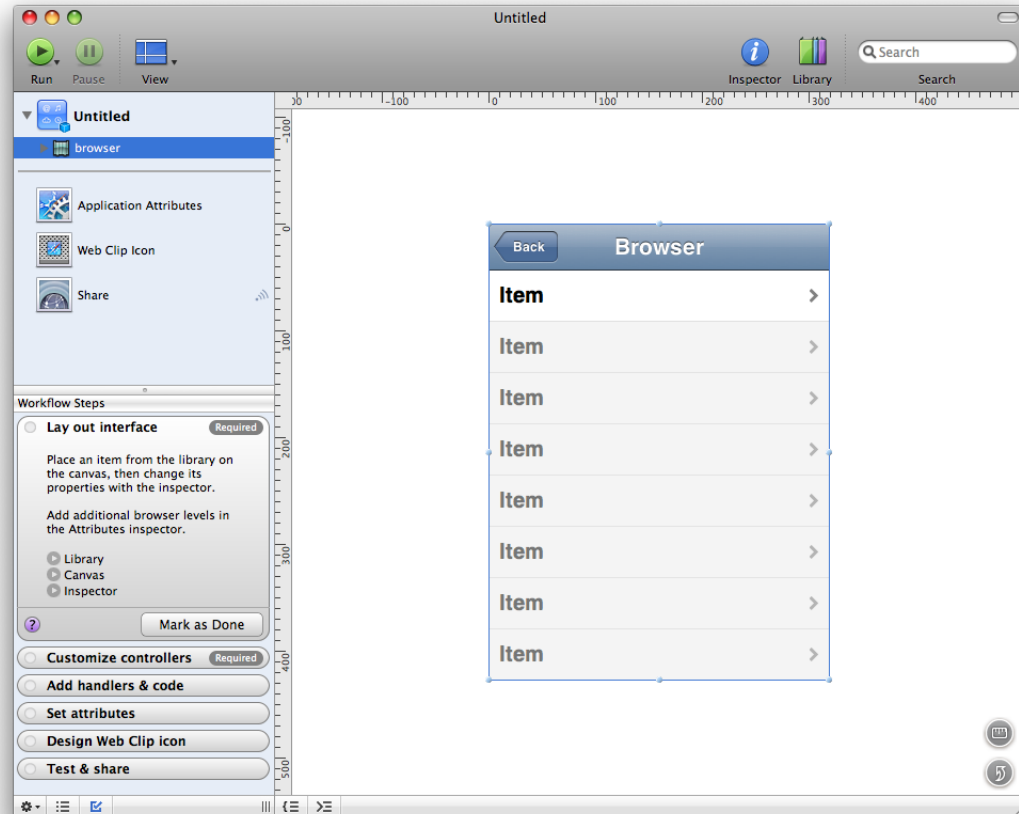
Before continuing, make sure that you have Dashcode version 2.0 or later installed on your Mac (the install location is `/Developer/Applications/`). If you don't have Dashcode installed, read "[Getting and Running Dashcode](#)" (page 10) to learn how to get and install Dashcode.

Choose a Template

To start, double-click the Dashcode icon to open it. A new project window opens and displays a dialog that displays two types of projects—Dashboard Widgets and Web Applications—and an assortment of templates for each project type. **Templates** are handy starting points for creating common types of web applications and widgets. (To find out what a template does, select it to show a short description of its capabilities.)

For this tutorial, select the Web Applications project type. Then, select the Browser template and click Choose. A project window opens, displaying the first page of a new web application based on the Browser template, as shown in Figure 2-1.

Figure 2-1 A new web application project based on the Browser template



Along the left side of the project window is the **navigator**, which you use to switch between the various tools available when you're designing a web application. The main portion of the window is the **canvas**, which you use to design your web application's interface.

At the bottom of the navigator in Figure 2-1 you can see the **Workflow Steps list**, which guides you through the web application development process. Each step is a milestone in creating a web application, telling you what to do and where to do it. When you complete a step, mark it as done and move on to the next step.

Note: If you don't want to see the Workflow Steps list, you can hide it by choosing View > Steps or by clicking the button that looks like a checkbox at the bottom edge of the project window (this button is highlighted in Figure 2-1).

Alternatively, you can view a list of the web application's files in place of the Workflow Steps list. If you want to see the Files list, choose View > Files or click the list button in the bottom edge of the project window (it's the button to the left of the checkbox button).

Learn About the Template

The Browser template gives you a web application that displays some built-in information and supports navigating from one page to the next. You don't need to customize the application at all to see how it works, but you should specify a title to display.

In the canvas, double-click Browser and type in a new title, such as Parks. You'll see this title in the header of the first page.

You should also give a name to the webpage itself. When you begin a project, this name is Untitled. To specify a name, double-click Untitled at the top of the navigator and type the name you want, say, National Parks.

Now is also a good time to save your project. Choose File > Save to save the project. Give your project a name and select a location to save it in. The project encapsulates the web application and information Dashcode needs to create the application for you.

Test the Application

Your new web application is already functional, even though it only displays placeholder data. To prove this, click the green Run button in the Dashcode toolbar to run the web application. (Alternatively, you can choose Debug > Run to test your application.)

Dashcode opens your web application in a simulator application. Although it does not look exactly as it would on iPhone or iPod touch, your web application runs so you can test it and fix any problems you encounter.

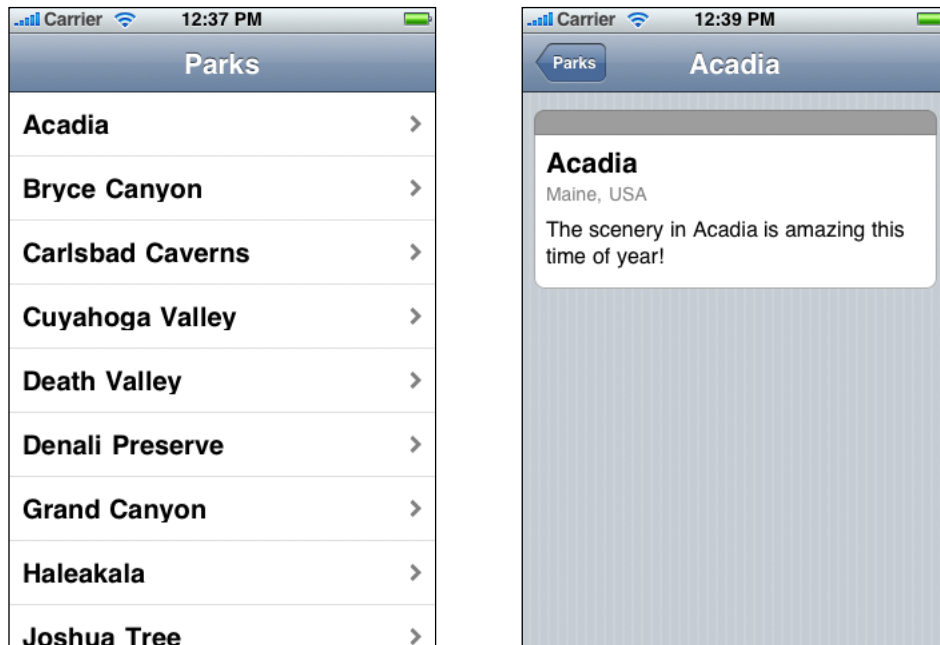
Take a few moments to use the application. You should see Parks (or a different title you typed in) displayed in the header of the first page and you should see National Parks (or a different title you gave the webpage) in the title bar area, if this area is visible.

Notice that when you click a row item in the first page, a new page appears that gives details about the item. There are a few other things you should notice about the second page:

- The title of the second page is the same as the name of the row item you selected in the first page.
- A back button has appeared to the left of the title and its label is Parks, which is the title of the first page. You can click this button to return to the first page.
- The text in the middle of the second page includes the name of the row item you selected.

The two pages of your web application should look similar to those shown in Figure 2-2.

Figure 2-2 The default Browser-based web application contains a top-level page (left) and a detail page (right)



Although the information about national parks is just placeholder information provided by the Browser template, it helps you see some of the connections between the pages. You'll learn a bit more about these connections in the next step, [“Explore the Views in the Stack Layout”](#) (page 22).

Before you continue, quit the simulation by choosing Quit from the simulation application menu or pressing Command-Q. Or, in Dashcode, you can click the red Stop button.

Explore the Views in the Stack Layout

A browser-style web application allows users to navigate from one page to the next. As you found out when you tested it, your web application already supports the ability to select an item in the top-level list and reveal information about it in a detail view. One of the keys to this structure is the stack layout.

If you can't already see it in the navigator, reveal the stack layout by clicking the disclosure triangles next to National Parks and, below that, next to `browser`. When you click the disclosure triangle next to `stackLayout`, you see two items: `listLevel` and `detailLevel`. In this application, the list level view contains the list of parks, and the detail level view contains the text that describes a selected park.

You can change the way the detail page is revealed when you select a row in the top-level list. To do this, follow these steps:

1. Make sure the inspector is open. If it isn't, click the blue Info button in Dashcode's toolbar.
2. Select `stackLayout` in the navigator.
3. Click the Attributes toolbar item in the inspector (it's the one on the left). This should change the inspector's title to Attributes (stackLayout).

At the bottom of the inspector is a section titled Subview Transition. The values in this section control the transitions between pages. You can choose to have pages slide in from right to left (this is the default) or other ways, such as from top to bottom. Note that you may not be able to see the effect of changing the transition type in the simulator application, but you will see the effect when you run your application on an actual device.

Note: The transitions described in this step are based on CSS transitions and animations. In the stack layout Attributes inspector, Dashcode gives you a way to specify attributes of the transitions you want to use, but does not provide a user interface for manipulating them directly. After you complete this tutorial, you might want to access the code for these transitions to make finer-grained adjustments. Before you do this, you should read *Safari CSS Animation Guide for iPhone OS* and *Safari CSS Transform Guide for iPhone OS* for more information.

Add Functionality Using Parts

As you've seen, the web application you created using the Browser template is already functional even though all its data is static. To make it more useful, you can add functionality that gets current information from the web. In this step, you use Dashcode's design tools to add a button to the detail page that performs a Google search on the featured park (you'll add the code to support this action in the next step).

To add a button to your web application, use a button **part**. Parts are controls and views used in a web application's user interface. Dashcode lists the available parts in the Library.

To find the appropriate button part, choose Window > Show Library and click Parts. Scroll through the list of parts until you find the push button part, and drag this part into your detail page on the canvas.

Double-click the button on your detail page and give it a label (you can also do this by typing the label in the Label field of the Attributes inspector for the button). You may need to resize the button to fit the label you choose. Figure 2-3 shows the new button in the detail view, before it receives a new label.

Figure 2-3 Adding a part to the web application's user interface



Write Code to Perform a Search

To make the button perform a search on the featured park, you need to add behavior to the button. In the Inspector, click Behaviors (it's the rightmost button). This shows the Behaviors pane, in which you assign handler functions to events on an object.

On the canvas, select the button you added to the detail page. In the Behaviors inspector, double-click in the Handlers column next to the `onclick` event name. Enter the name of a new function, such as `detailButtonHandler`, and press Return. Click the arrow next to the function name you entered to reveal the **source code editor** below the canvas. Clicking the arrow reveals the `detailButtonHandler` function Dashcode inserted in your web application's code.

Between the braces (`{ . . . }`), enter the following line of code:

```
document.location = ("http://www.google.com/search?client=googlet&q=" +
detailController._representedObject);
```

This line of code tells the web application to display the search results for the featured park (`detailController._representedObject` contains the name of the park on the detail page). After you add this functionality, the code in the source code editor should look like Figure 2-4.

Figure 2-4 Adding code to handle the button's click event

Test your web application again by clicking Run (or by choosing Debug > Run). In the detail page for a park, click the button you added and make sure it displays the results of a Google search on the featured park (be sure you're connected to the internet before you try this).

Deploy Your Web Application

Congratulations! You've created your first web application using Dashcode.

To use your application on an iPhone or iPod touch, you need to deploy the application and make it available on a web server. To learn how to do this, see ["Deploying a Web Application"](#) (page 48).

Starting a Project

When you start a new project, you base it on a Dashcode template or, in the case of widgets, you can also base it on an existing widget. Widget projects encapsulate all the resources that Dashcode uses to build a widget for you. Similarly, web application projects contain all the resources Dashcode needs to build a web application.

This chapter discusses the options you have when you start a project. It also discusses opening a widget in Dashcode, a feature that allows you to forgo Dashcode's design tools to code, test, and debug an existing widget.

Creating a Project from a Template

Dashcode creates new projects based on templates. Templates are preconfigured widgets or web applications that include code and graphics that perform common tasks. When you open Dashcode or choose File > New Project after you open it, a dialog appears that offers you a choice of widget or web application project types. When you choose one of these, the dialog offers you a choice of templates appropriate for the project type. Click a template icon to see a short description of the template's abilities. If the template matches the task you're trying to perform, select its icon and click Choose. For more on the templates included with Dashcode, read "[Dashcode Templates](#)" (page 53).

After you choose a template, you may need to provide required values for the widget or web application to work properly. For widgets in particular, make sure to provide the Identity and Properties values, as discussed in "[Providing Widget Attributes](#)" (page 28).

Creating a Project from an Existing Widget

Dashcode can create a new project from an existing widget. If you want to continue work on an existing widget in Dashcode, you should import it. Importing a widget copies an existing widget into a new widget project and enables the code generator. When the code generator is active, all of Dashcode's design and management tools are enabled.

To import a widget, choose File > Import Widget or choose Import from the template chooser dialog.

Note: If your widget uses objects created at runtime (such as those created from an Apple Button or an Apple Scroll Area, both Apple classes that Apple provides widget developers), they do not appear on the canvas since their constructors are not executed when the widget is shown.

Once you import your widget into a Dashcode project, use the widget attributes pane, as discussed in "[Providing Widget Attributes](#)" (page 28), to edit your widget's Identity values.

Providing Widget Attributes

Each widget project has required values you need to provide for the widget to work properly. You set these values in the widget attributes pane, available when you select Widget Attributes in the navigator. Make sure to provide values in the following sections of the editor, if available:

Identity

The values in this section are used to identify a widget.

You need to provide a widget identifier, used by Dashboard to differentiate your widget from others. Widget identifiers are commonly formatted in reverse domain notation, starting with a top-level domain (such as `com`), followed by a company or creator name (such as `apple`), and then a unique product name (such as `my-fabulous-widget`, yielding a name such as `com.apple.my-fabulous-widget`).

Additionally, you need to provide a unique version number. This number is used by Dashboard to differentiate between versions of a widget, so that it's always running the most recent version.

Note: The widget identifier and version fields correspond to the `CFBundleIdentifier` and `CFBundleVersion` information property list values, as discussed in [Dashboard Info.plist Keys](#).

Properties

Template-specific options are in this section. If you're using an imported or opened widget or the Custom template, this section is absent. For each template's specific properties, read [“Dashcode Templates”](#) (page 53).

Providing Attributes for a Web Application

Each web application project has values you need to provide for the application to work properly. You set these values in the web application attributes pane, available when you select Application Attributes in the navigator. Although only the page title is required, it's a good idea to set the other values so your web application behaves as you intend. Provide values in the following sections of the editor, if available:

General

The value in this section is used to identify the web application.

In the Page Title field, title your web application by giving it an appropriate, human-readable name. The title you supply is the `<title>` element in your webpage and is displayed in the title bar of the browser.

Viewport

The values in this section control how users can view your web application when they use it on iPhone or iPod touch. When users change the device orientation from portrait to landscape, webpages can get scaled to fit the new screen orientation.

The two options for the Orientation value are:

- “Adjust page width to fit.” When you choose this option, your web application resizes (that is, increases or decreases in width) when the device orientation changes. This option is generally recommended for web applications, because it enhances the user's perception of the web application as a standalone application and not a webpage.

- "Zoom pages to fit." When you choose this option, the width of your web application does not change when the device orientation changes, but the scale does. In other words, the content of your web application will appear a bit bigger. You might want to choose this option if your web application has a complicated layout that you don't want to change in width when the device orientation changes.

The Page Zooming value in this section controls whether users can zoom your web application when they view it on iPhone or iPod touch. In general, because you want iPhone and iPod touch users to view your web application as a standalone application and not as a webpage, it's recommended that you turn off Page Zooming (that is, deselect "Allow users to adjust page zoom").

Web Clip

The values in this section control how your web application can be displayed on iPhone and iPod touch and how your Web Clip icon is created and displayed.

In iPhone OS 2.0 and later you can choose to make your web application available in a full-screen mode, which hides the Safari toolbar and navigation bar. If you want to do this, you must also provide a Web Clip icon, because tapping a Web Clip icon is the only way users can open a web application in full-screen mode (navigating to the application in Safari does not open it in full-screen mode). If you choose not to support full-screen mode, you should still consider providing a Web Clip icon so users have a convenient way to open your web application.

To make your web application available in full-screen mode, select the "Show as full screen application (hide Safari toolbar and navigation bar)" checkbox. This allows users to open your web application in full-screen mode when they tap the Web Clip icon. Note that users press the Home button to leave a web application that is running in full-screen mode.

If you selected the "Show as full screen application (hide Safari toolbar and navigation bar)" checkbox, you can also specify one of the following three styles for the status bar:

- Gray
- Black
- Black (Translucent)

Use the pop-up menu to choose the status bar style that best coordinates with the appearance of your web application. You can specify a status bar style only if you selected the "Show as full screen application (hide Safari toolbar and navigation bar)" checkbox. This is because the status bar appearance automatically matches the appearance of the Safari navigation bar when it is visible.

The two options for the Icon value are:

- "Use custom icon." Choose this option if you want to design your own Web Clip icon to represent your web application. See "[Designing a Web Clip Icon for a Web Application](#)" (page 36) for more information on how to do this.

If you choose this option, you can also specify whether you want the shiny glass overlay effect to be added automatically by selecting the "Add glass visual effect" checkbox. Note that you do not have control over a Web Clip icon's rounded corners and drop shadow; these are always added automatically.

- "Use Safari generated icon." Choose this option if you want to allow users to create their own Web Clip icon to represent your web application.

Opening an Existing Widget

If you want to forgo Dashcode's project management and design tools, you can open an existing widget in Dashcode without importing it into a new project. When Dashcode opens a widget, you have access to its code and all of Dashcode's debugging tools, but not its design tools.

To open a widget in Dashcode, choose File > Open and select a widget in the Open dialog, or choose File > New Project and click the Open Existing button at the bottom of the template chooser dialog. When you open a widget with Dashcode, the canvas is locked. This means that Dashcode's design tools, responsible for generating HTML, CSS, and images for a widget, are turned off. When you open an existing widget instead of importing it into a new project, Dashcode displays a lock over the bottom of the canvas.

Designing a Widget or a Web Application

After you've started a project as discussed in [“Starting a Project”](#) (page 27), use Dashcode's design tools to design the interface of the widget or web application. This chapter describes the canvas, the interface design area in Dashcode, and how to begin laying out an interface. It also discusses how to add Dashcode parts to your project and how to use the inspector window to adjust the appearance or behavior of a part.

Read this chapter to learn how to use Dashcode's design tools to create, modify, and preview the appearance of your widget or web application.

Laying Out the Interface

Dashcode includes a workspace where you lay out the elements that comprise the user interface of a web application or a widget. This workspace, called the **canvas**, is visible in the main portion of the project window when you select your widget or web application in the navigator. You can freely move and resize any element in an interface when the canvas is visible. To move an element, drag it to where you want it to be. To resize an element, drag one of its resize handles. If you hold down the Shift key while resizing an element, the original proportions are maintained.

Dashcode also provides commands you can use to arrange and align elements in a widget or web application interface. For more on how to do this, see [“Arranging and Locking Elements”](#) (page 33).

You can drag any item from the Finder to the canvas to add it to a your widget or web application. This means that you can design the user interface in an application such as Adobe Illustrator or Photoshop, save the images, and drag them into your Dashcode project from the Finder. Dashcode, however, offers its own design elements. Learn about them in [“Adding Parts to an Interface”](#) (page 32).

Showing a Widget's Sides

By default, all new widgets created from a Dashcode template have two sides: a front and a back. The canvas shows you only one of these sides at a time. To switch between the sides, click front or back in the navigator. (If you don't see front and back in the navigator, click the disclosure triangle next to the widget's name.)

When a side is visible on the canvas, you can add and arrange elements on it. To work on the other side, click the corresponding name in the navigator.

Adding Parts to an Interface

Dashcode includes a set of controls, shapes, and views called **parts**. Choose Window > Show Library and click the Parts button to see the parts included with Dashcode. Some parts, such as the Go Back and Go Forward buttons, are intended for use in web applications.

To add a part to a widget's or web application's interface, drag it from the Parts Library to the widget body or web application page on the canvas. Once a part is on the canvas, you can change its properties, as discussed in ["Changing an Element's Properties"](#) (page 32). If you want to know more about laying out parts in your project's user interface, read ["Arranging and Locking Elements"](#) (page 33).

A number of Dashcode parts can be manipulated programmatically. Read ["Dashcode Parts"](#) (page 59) to learn more about using these parts.

Using Photos from iPhoto

You can drag any photo in your iPhoto library to the canvas to include it in a widget or web application. To show your iPhoto library, choose Window > Show Library and click the Photos button. Once the photo is on the canvas, you can arrange it as described in ["Arranging and Locking Elements"](#) (page 33) and adjust its properties as discussed in ["Changing an Element's Properties"](#) (page 32).

Note: The iPhoto library requires iPhoto 6 or later.

Changing an Element's Properties

The inspector window reveals the attributes for a selected element on the canvas. In this window, you can view and edit the element's properties based on the inspector type you choose. Choose an inspector by clicking its button at the top of the window. The inspector window includes the following five inspectors, from left to right:

- The Attributes inspector allows you to modify an element's ID (used in JavaScript to reference the element), its CSS class, whether it's shown in the widget or web application and in the default image, and any parameters that are unique to the element.
- If it's appropriate for the selected element, the Fill & Stroke inspector allows you to adjust a shape's or control's fill style, opacity, corner roundness, and stroke style.
- The Metrics inspector allows you to modify an element's size and position, as well as its behavior if the widget or webpage is resized, as controlled by the Autoresize and Constraints sections. The Autoresize settings affect how an element behaves when the widget or webpage is resized on the canvas.
- If the selected element displays text, you can adjust the text's font, style, color, size, shadow, alignment, and spacing in the Text inspector. You can also set whether text wraps or not and how to handle text overflow. Setting the text overflow to Clip cuts any string in the selected element at its bounds, allowing for no overflow, whereas setting the text overflow to Ellipsis appends an ellipsis character (...) to the selected element's text when it reaches the element's bounds.

Note: If you plan to share a widget or deploy a web application, be careful to use fonts that are standard in Mac OS X, such as Helvetica Neue, Times, and Monaco.

- The Behaviors inspector allows you to assign to an element JavaScript handlers for various events. For each event, you can assign an existing JavaScript function as its handler or create a empty function that's automatically added to the project's JavaScript code. Once you assign a handler to an event, click the arrow next to the handler's name to reveal the function in the source code editor.

Working with code is discussed in [“Adding Source Code”](#) (page 39).

Arranging and Locking Elements

Dashcode offers helpful options for arranging elements on the canvas. If an element is obscured by other elements, select it and choose `Arrange > Bring Forward` or `Arrange > Bring to Front` to move it in front of any elements currently obscuring it. Similarly, if you want to move an element behind another, select it and choose `Arrange > Send Backward` or `Arrange > Send to Back`.

Dashcode offers alignment and distribution options for arranging multiple elements with respect to each other. If you want to align a few items on their left edges, you select them on the canvas and choose `Arrange > Align > Left`. Similar options are available to align elements by center, right, top, middle, and bottom. If you want an equal amount of vertical space between multiple elements, choose `Arrange > Distribute > Vertically`. A similar option exists to distribute elements horizontally.

If you want to lock an element so that you can no longer move it, select the element and choose `Arrange > Lock`. If you change your mind and want to move it again, select it and choose `Arrange > Unlock`. Note that locking an element only locks its placement on the canvas—you can still adjust its attributes in the inspector.

Absolute versus Document-Flow Positioning

Dashcode allows you to position elements in your widget or web application using either absolute or document-flow (that is, relative) positioning. You can change the positioning of a selected element by switching between Absolute and Document Flow in the Layout section of the Metrics inspector.

Briefly, absolute positioning means that an element's position within its containing element is always the same, regardless of the positions of its sibling elements. In other words, an absolutely positioned element retains its position within the containing element, even if the sibling elements around it expand or shrink. Absolute positioning allows you to have precise control over complicated layouts, but does not support dynamic resizing very well.

In contrast, document-flow positioning means that an element's position is defined in terms of the positions of its document-flow sibling elements. In document-flow positioning sibling elements are displayed in order, vertically, so the position of an individual element depends on its location in the series of elements. If one element expands downward, it pushes down all the sibling elements that come after it by the same amount, without changing their sizes or their relative positions. Although document-flow positioning provides great flexibility to support dynamic resizing, it does not give you fine-grained control over layouts.

Because widgets tend to stay the same size (or switch among a small number of predefined sizes), Dashcode widget projects use absolute positioning by default. Conversely, because web applications need more flexibility in sizing, Dashcode web application projects use document-flow positioning for most elements by default.

The default positioning style used in a project does not mean that there are no variations, however. In the Browser template web application project, for example, the elements in the list row (the row title and the row arrow) are absolutely positioned to ensure a consistent layout.

When you customize a project, you should be aware of the positioning style of the element you're changing. This is because dragging a part into an existing container element on the canvas causes the part to adopt the prevailing positioning style of its new siblings. If you want the new part to adopt the alternate positioning style, hold down the Shift key while you drag the part onto the canvas.

Searching For Elements

Use the search field in the toolbar to find elements in your project's interface and code. This particular type of search works when you're designing and coding a widget or web application. When Dashcode is running a widget, the toolbar's search field narrows results from either the run log or the Stackframe & Variables table, depending on which is visible. This is discussed further in [“The Run Log and Tracing Execution”](#) (page 46) and [“Checking Values in Memory”](#) (page 47).

To search for an element, type part of the element's name in the search field. When the search results are returned, the navigator is replaced with search results. Selecting an item in the search results either highlights it on the canvas or shows the source code file that contains the implementation of the element.

Rulers and Guides

By default, the canvas displays rulers along its top and left sides. Rulers, when used with guides, help you align elements relative to one another. To add a guide, position the pointer anywhere in a ruler and drag towards the canvas. As you drag away from the ruler, a guide appears—it remains wherever you stop dragging. To remove a guide, drag it back to a ruler. To hide the rulers, choose View > Hide Rulers.

Placing the Close Box

Users click a widget's close box to close it (web applications do not have a close box). By default, a widget's close box appears over the top left of a widget's body. If necessary, move the close box so that it overlaps the top left corner of your widget's body. To hide the close box on the canvas, choose View > Hide Invisible Items.

Disabling the Canvas

Because the canvas generates HTML and CSS automatically for you, you may want to turn its code generation off if you're tweaking elements by hand. To turn off the automatic code generator, choose **View > Stop Code Generator**. When you're finished tweaking values by hand, you can turn the code generator back on by choosing **View > Start Code Generator**.

Previewing a Widget's Default Image

The default image preview shows you a widget's default image, which is displayed while it loads in Dashboard. To see your widget's default image, select **Default Image** in the navigator.

By default, all the elements on the front of a widget except the text parts (**Text**, **Text area**, and **Text field**) are included in a widget's default image. Other parts that display text, such as pop-up menus, are also included. You are discouraged from leaving text in your widget's default image because any text on your interface may change when the widget is localized. Therefore, you should remove parts that display text from your widget's default image. To remove any element from your widget's default image, follow these steps:

1. Select the widget item in the navigator to show the canvas.
2. Select the element on the canvas that you want removed from the default image.
3. Show the **Attributes inspector** (as discussed in [“Changing an Element's Properties”](#) (page 32)).
4. Deselect **Show in Default Image**.

The toolbar below the default image preview offers additional configuration options for default images:

Start Sync / Stop Sync

By default, Dashcode updates a widget's default image whenever the widget's interface changes. To turn this behavior off, click **Stop Sync**. To enable this behavior if it's been turned off, click **Start Sync**.

Import

If you already have an image that you want to use as a widget's default image, click the **Import** button and select it in the dialog that appears.

Open in External Editor

If you want to tweak your default image in an application other than Dashcode, click the **Open in External Editor** button. When you do, the default image as shown in Dashcode is opened in your default PNG-handling application.

Designing a Widget Icon

You can use the widget icon editor to design a widget's icon. The widget icon represents a widget in the Dashboard widget bar and in the widget manager. To show the widget icon editor, select **Widget Icon** in the navigator.

To modify the appearance of the body of the widget icon, show the Fill & Stroke inspector and change the fill style, corner roundness, opacity, and stroke style. The toolbar at the bottom of the widget icon editor offers additional configuration options for widget icons:

Start Sync / Stop Sync

By default, Dashcode syncs the style of a widget's front image with its widget icon. To turn this behavior off, click Stop Sync. To enable this behavior if it's been turned off, click Start Sync.

Place

To put an image, such as a logo, on a widget's icon, click the Place button and select the image file. Once the image is placed, you can move and resize it.

Import

If you already have an image that you want to use as a widget's icon, click the Import button and select it in the dialog that appears.

Open in External Editor

If you want to customize a widget icon in an application other than Dashcode, click the Open in External Editor button. When you do this, the icon is opened in your default PNG-handling application.

Designing a Web Clip Icon for a Web Application

A web application can provide a Web Clip icon for users to place on their Home screens and use as a type of bookmark for the application. When a user taps a Web Clip icon the web application opens automatically, without requiring the user to navigate to it. Web Clip icons should be simple, attractive, and easy for users to recognize.

Providing a Web Clip icon is a good idea for most web applications, but for web applications that can run in full-screen mode, it's essential. This is because the only way users can experience a web application's full-screen mode is by tapping its Web Clip icon to open it. For more information on specifying the mode your web application can run in, see ["Providing Attributes for a Web Application"](#) (page 28).

Note: Because iPhone and iPod touch add the appropriate corner radius and drop shadow to the icon you provide, you should not add these effects yourself. However, you can control whether the shiny glass overlay effect is added automatically by specifying a value in the Application Attributes pane (for more information about this, see ["Providing Attributes for a Web Application"](#) (page 28)).

You can use the Web Clip Icon composer to design your application's icon. To show the Web Clip Icon composer, select Web Clip Icon in the navigator.

To modify the appearance of the body of the icon, show the Fill & Stroke inspector and change the fill style, opacity, and stroke style. The toolbar at the bottom of the Web Clip Icon composer offers additional configuration options for these icons:

Place

To put an image, such as a logo, on a web application's icon, click the Place button and select the image file. Once the image is placed, you can move and resize it.

Import

If you already have an image that you want to use as a web application's icon, click the Import button and select it in the dialog that appears.

Open in External Editor

If you want to customize a web application icon in an application other than Dashcode, click the Open in External Editor button. When you do this, the icon is opened in your default PNG-handling application.

Adding Source Code

After you've started a project and designed the user interface, you may need to add code to your widget or web application. This chapter describes how to view the files that make up your project and add code to them to perform tasks that aren't automatically provided by the template. It also shows you how to turn on access to resources, such as files outside of the project and the network. Finally, it covers the application preferences that affect your code editing experience in Dashcode.

Viewing a Project's Source Code

Dashcode includes all the HTML, CSS, and JavaScript files that constitute a widget or web application. When you want to add functionality beyond what a template provides, you need to view and edit these implementation files. To show your project's implementation files, choose View > Files. This reveals the Files list under the navigator. In the Files list, you can add, duplicate, and rename files and folders, as discussed in ["Adding and Moving Files and Folders"](#) (page 41).

Note: When you edit a file's name, make sure that you make the change in all references to that file. File references are usually in the `Info.plist` and HTML files. Be sure to review these files and, if there are references to old filenames, update them to the new name.

When you select an HTML, CSS, JavaScript, or property list file in the Files list, the **source code editor** appears under the canvas showing the selected file's contents. In the code editor you can edit the actual HTML, CSS, and JavaScript files that implement your widget or web application. For more on coding using HTML, CSS, and JavaScript, read ["Using HTML, CSS, and JavaScript Programming Interfaces"](#) (page 40).

Above the text view portion of the source code editor are two pop-up menus, a file pop-up menu on the left and a function pop-up menu to the right of that. The file pop-up menu lists all the text files in a widget or web application that you can edit using the source code editor. When you select a file in the file pop-up menu, it's displayed in the source code editor. The function pop-up menu lists the names of all the functions in a JavaScript file or all the classes in a CSS file. The function pop-up menu is not available when viewing HTML and property list files.

A common way to add code is to add an event handler to an element in the user interface. You do this using the Behaviors inspector, as discussed in ["Adding a Handler for an Event"](#) (page 40).

When working with resources originating outside a widget or web application—XMLHttpRequest, Command-line tools, Java applets, and such—you need to explicitly turn on access to these items. To learn more about resource access and what elements need activation, read ["Resource Access"](#) (page 42).

You can also modify how the source code editor looks and behaves, as described in ["Code Editing Preferences"](#) (page 43).

Using HTML, CSS, and JavaScript Programming Interfaces

After you've revealed a project's source code, you can modify its HTML, CSS, and JavaScript files, as discussed in [“Viewing a Project's Source Code”](#) (page 39). Any HTML, CSS, or JavaScript code that works in Safari and its WebKit engine can be used in a widget or web application. To learn more about HTML, CSS, and JavaScript, read these documents:

- [Safari FAQ](#)
- [WebKit DOM Programming Topics](#)
- [Safari HTML Reference](#)
- [Safari Web Content Guide for iPhone OS](#)
- [WebKit DOM Reference](#)

To learn more about creating a webpage or web application to run in an iPhone or iPod touch, see *Safari Web Content Guide for iPhone OS*.

In addition to WebKit's HTML, CSS, and JavaScript programming interfaces, Apple offers Dashboard-specific programming interfaces for use in widgets. For more on these programming interfaces, read *Dashboard Programming Topics* and *Dashboard Reference*.

Adding a Handler for an Event

When you add a handler using the Behaviors inspector (as discussed in [“Changing an Element's Properties”](#) (page 32)), the new handler function is inserted in the project's JavaScript file. When you click the arrow in the Behaviors inspector next to the handler function's name, the source code editor appears under the canvas with the function selected. You can then add your custom code to handle the event.

For more on using the source code editor, read [“Viewing a Project's Source Code”](#) (page 39). To learn about JavaScript and the Dashboard programming interfaces, read [“Using HTML, CSS, and JavaScript Programming Interfaces”](#) (page 40).

Adding Code for Custom Controllers

If you want to modify a web application by changing or adding views, such as list or detail views, you also need to add code to control them. Each view in a web application is controlled by a JavaScript controller object. For views that contain lists, the controller object serves as the data source for the list. More generally, the controller object is in charge of configuring the interface within a view so that it displays properly, and so that the view can handle all activity that occurs within it.

For more information on some of the parts you can add to your web application, see [“Dashcode Parts”](#) (page 59). To learn about JavaScript and the Dashboard programming interfaces, read [“Using HTML, CSS, and JavaScript Programming Interfaces”](#) (page 40).

Code Completion Using Code Sense

Dashcode's source code editor includes a code completion feature that suggests possible variable and function names based on partially entered text. When Code Sense has a suggestion, it underlines the text you're typing. To see the list of suggestions, press the Option and Escape keys. Use the Up and Down Arrow keys to select the symbol you want. To add the symbol, press the Tab key. To dismiss the suggestion list, press the Escape key.

See [“Code Editing Preferences”](#) (page 43) for more information on how to set Code Sense preferences.

Code Snippets

Dashcode includes a set of reusable **code snippets**. Each code snippet provides a bit of functionality that you can use to enhance your widget or web application. To show the code snippets, choose Window > Show Library and click the Code button. To add a code snippet to your project, drag it from the Code Library to the source code editor.

In addition to the code snippets included with Dashcode, you can save your own snippets for later use by dragging them from the source code editor to a group you create in the Code Library. To create a new group, choose New Group from the action menu under the listings in the Code Library.

Adding and Moving Files and Folders

You can add additional files and folders to a widget or web application in Dashcode. Although this is rarely necessary, it can be helpful when organizing numerous files or areas of functionality. To add a new file, show the files list, choose File > New and choose either JavaScript File, CSS File, or File. After you create a new file, you can view and edit its contents as described in [“Viewing a Project's Source Code”](#) (page 39).

To add a folder to your project, choose File > New > Folder. To add items to the folder, drag them on to its icon.

In the Files list, you can rename, duplicate, move, and delete files and folders. To rename an item, select it in the Files list and choose Rename from the action menu (the gear icon beneath the Files list). To duplicate an item, choose Duplicate from the action menu. To remove an item, choose Move to Trash from the action menu. To move an item, drag its icon to another folder's icon.

Note: Dashcode does not manage references to files and folders in a widget or web application. If you remove, rename, or move a file or folder, you need to remove, rename, or update the reference to the file or folder in the project's main HTML file. Failure to update a reference to a removed, renamed, or moved file or folder may result in unexpected behavior.

When you add a new JavaScript or CSS file, you need to include a reference to the file for it to be used by your widget or web application. Add a reference to the new file at the top of the project's main HTML file. In a widget, the main HTML file is the file paired with the `MainHTML` key in the widget's `Info.plist` file. In a web application, the main HTML file is the `index.html` file.

In your project's main HTML file, add a reference to a new CSS file using the `@import` directive using the `<style>` tags. Add a reference to a new JavaScript file using the `<script>` tag.

Resource Access

If you intend to use any of the following resources in your widget, you need to turn on access to them first:

- Network resources (including `XMLHttpRequest`)
- External files
- Internet plug-ins, such as QuickTime and Quartz Composer
- Java applets
- Command-line applications

Important: In general, Safari on iPhone does not support any third-party plug-ins or features that require access to the file system, so most of these resources are not appropriate for web applications intended for use on iPhone or iPod touch.

However, Safari on iPhone does support `XMLHttpRequest`. To learn more about unsupported technologies in an iPhone web application, see *Safari Web Content Guide for iPhone OS*.

To turn on access to these resources, select **Widget Attributes** in the navigator and select the option for the resource you're trying to use. Options include:

Network / Disk Access

If your widget requires access to network resources or files on disk, select the appropriate item. Unless your template already needs access to network resources (such as `XMLHttpRequest`) and files on disk outside of your widget bundle, these resources are turned off.

Extensions

If your widget uses content provided through an Internet plug-in or a Java applet, or uses a command-line utility via `widget.system`, select the appropriate option. Unless your template already needs access to plug-ins (such as QuickTime), Java, and command-line utilities, these resources are turned off.

Code Editing Preferences

The source code editor has a number of preferences you can set to change its behavior to better suit your tastes. To show these preferences, choose Dashcode > Preferences. The Preferences window includes the following items:

- General preferences include an option for opening source code files in Dashcode or in a helper application.
- The source code editor has various visibility and behavior preferences, including the visibility of the gutter and line number next to your source code, the source code editor's line wrapping, and the Tab key's indentation behavior. You can set these preferences in Editing preferences.
- Your source code can be colored based on the code's syntax. You can adjust text font, size, and color in Formatting preferences.
- Dashcode offers a code completion feature called Code Sense (as described in [“Code Completion Using Code Sense”](#) (page 41)). At the top of the source code editor, a pop-up menu shows the symbols in a file. Code Sense preferences include a setting for how these symbols are organized. This pane also includes settings for code completion.

Testing and Sharing

After you've written code for your widget or web application, test it to ensure that it works as you expect. Dashcode includes its own Dashboard environment so you can run and test a widget in Dashcode without having to open it in Dashboard. Similarly, Dashcode includes a simulator application that mimics some of the features of iPhone and iPod touch, so you can easily test a web application.

In addition to running widgets and web applications, Dashcode also includes tools that help you debug and fix problems that may arise. This chapter includes information on how to trace events and inspect variable values while a widget or web application executes. It also includes information on sharing a widget and deploying a web application.

Testing a Widget or a Web Application

To run a widget or a web application in Dashcode, choose **Debug > Run** or click **Run** in the toolbar. Depending on your project type, this has a different effect:

- For widgets, this runs the widget in Dashcode without having to open it in Dashboard. The widget behaves here as it would in Dashboard.
- For web applications, this runs the web application in a simulator application without having to deploy it and run it in a device. The simulator allows your web application to behave as it would in a device.

In addition, you have two options when testing a web application in Dashcode version 2.0.1 and later:

- Run the web application with the Safari interface visible (that is, with the Safari toolbar and navigation bar visible). This simulates what users can see when they use Safari to navigate to your web application.
- Run the web application with the Safari interface hidden (that is, in full-screen mode). This simulates what users can see when they tap a Web Clip icon to open a web application that can run in full-screen mode.

You can access these options by clicking and holding the **Run** button in the Dashcode project window. From the menu that appears, choose **"Run"** or **"Run Full-screen."** After you make your choice, clicking **Run** or choosing **Debug > Run** runs your web application in that mode until you change it.

When your widget or web application loads, test it out to see if it functions as you expect. When an error occurs, an **exception** is thrown and, by default, execution pauses. When this happens, there are a number of tools at your disposal that help you see where the problem is and inspect variable values:

- The run log lists errors and other useful information. If you enable tracing, the run log notes the invocation of functions as well. See ["The Run Log and Tracing Execution"](#) (page 46) for details.
- The Stackframe & Variables table shows you the value of any variable a widget or web application uses. See ["Checking Values in Memory"](#) (page 47) for details.

- Step-by-step execution enables walking through the execution line-by-line so you can see the effects of the code. See [“Pausing and Step-by-Step Execution”](#) (page 46) for details.
- The Evaluator is a console where you enter individual lines of code to be executed immediately. See [“The Code Evaluator”](#) (page 48) for details.

To continue a paused widget or web application, choose `Debug > Continue` or click `Continue` in the toolbar. To turn off pausing when an exception occurs, deselect `Debug > Break on Exceptions`.

To pause a widget or web application at any time, choose `Debug > Pause` or click the pause button in the toolbar. While a widget or web application is paused, you can step through its code line-by-line as discussed in [“Pausing and Step-by-Step Execution”](#) (page 46). You can also pause a widget or web application when execution reaches a specific line of code using a breakpoint, as discussed in [“Breakpoints”](#) (page 47).

To stop running a widget or web application in its testing environment, choose `Debug > Stop` or click `Stop` in the toolbar.

The Run Log and Tracing Execution

When you run a widget or web application in Dashcode, the canvas is replaced by the run log. Any errors that Dashcode encounters while running the widget or web application are reported here. For example, a call to a function that doesn't exist appears in the run log as an “object that doesn't allow calls.”

If the run log is replaced by another view, choose `View > Run Log` to show it. When possible, selecting a row in the run log reveals the associated line of code in the source code editor.

In addition to reporting errors, the run log can be used to trace execution of a widget or web application. Tracing means that an entry in the run log is made whenever a function starts and finishes. This is useful when you're watching the flow of execution in a widget or web application.

When the run log is visible, you can filter the contents of the run log by typing a term into the search field in the toolbar.

Pausing and Step-by-Step Execution

At any time, you can pause the execution of a widget or web application. Pausing execution is useful if you want to inspect the values of variables. To pause, choose `Debug > Pause` or click `Pause` in the toolbar. When the widget or web application is paused, the run log is replaced with the Stackframe & Variables table.

When the execution of a widget or web application is paused, you can inspect the values of the variables within the scope of the current function and the global scope. Also, the line of code that is paused is highlighted in the source code editor. In the shortcuts under the source code editor, you can see a hierarchy of functions that leads to the currently executing function. If you click another function name in the shortcuts, the function's variables are shown in the Stackframes & Variables table and its code is shown in the code editor. You can also search for a specific variable by typing its name into the search field in the toolbar.

When you're finished examining variables, you can continue executing the widget or web application in a few different ways:

Continue

Choose Debug > Continue or click Continue in the toolbar to continue execution with no interruption.

Step Into

Choose Debug > Step Into or click Step Into in the toolbar to execute the next line of code and step into function calls so you can see the effect that line of code has.

Step Over

Choose Debug > Step Over or click Step Over in the toolbar to execute the next line of code so you can see the effects that line of code has.

Step to End

Choose Debug > Step to End or click Step to End in the toolbar to execute the rest of the current function; execution pauses when the function is finished so that you can inspect its variables before they are relinquished.

Also, execution pauses whenever an exception occurs. When an exception occurs, the execution of a web application or widget is paused and an entry in the run log explains what the problem is. If you click the entry, the line of code in which the exception occurred is highlighted. By default, this option is enabled. You can control this option by choosing Debug > Break on Exceptions.

Checking Values in Memory

The Stackframe & Variables table shows the value of the variables used in a widget or web application. When the widget or web application is running, its global variables are available for inspection. When you pause a widget or web application, as described in [“Pausing and Step-by-Step Execution”](#) (page 46), the current function gets its own heading in the table with its variables listed underneath. If the function was called by another function, the second function’s variables are listed under the first, and so on.

Double-clicking a variable in the table adds the variable name to the code evaluator, as discussed in [“The Code Evaluator”](#) (page 48).

Breakpoints

In addition to pausing the execution of a widget or web application using the Pause option, you can set places in your code for execution to pause, called breakpoints. You can add a breakpoint in two ways. One way is to click in the gutter of the source code editor. If the gutter is not showing, go to Editing preferences and select the Show Gutter option.

A blue arrow in the gutter means that before that line of code is executed, execution will halt. To temporarily disable the breakpoint, click it; it turns from blue to gray to indicate that it’s disabled. To remove the breakpoint, drag it outside of the gutter.

The other way to set a breakpoint is to use the Breakpoints window. To show the Breakpoints window choose Debug > Show Breakpoints Window or double-click in the gutter. In the Breakpoints window, click the plus (+) button and specify either a filename and line number or a function name to break on. For instance, if you wanted to break in `Untitled.js` on line 42, you enter `Untitled.js:42`. Alternatively, you can supply the name of the function you want to break on (for instance, `showFront` in a widget).

In addition to the breakpoint, you can set a condition for the breakpoint. A condition is a JavaScript statement that evaluates to either `true` or `false`. If the condition evaluates to `true` when execution passes the breakpoint, execution pauses.

In the Breakpoints Window, you can remove a breakpoint by selecting a breakpoint from the table and clicking the minus (-) button or you can disable it by deselecting the checkbox next to a breakpoint's item in the table.

The Code Evaluator

While you run a widget or a web application, it can be useful to execute just one line of code. The code evaluator lets you do this. To show the code evaluator, choose `View > Evaluator` or choose `Evaluator` from the `View` menu in the toolbar. In the evaluator, you enter arbitrary code and press the `Return` key to execute the code immediately.

If you double-click a value in the `Stackframes & Variables` table, its name is automatically entered into the code evaluator. Also, if your cursor is at the prompt and you press the `up` arrow key, you cycle through the history of entries in the code evaluator.

Sharing a Widget

When you're finished developing a widget, you can share it with others. There are two commands for deploying a widget, and both export the widget from the project as a `.wdgt` bundle, ready to run in Dashboard. When you click `Share` in the navigator, the share pane gives you the choice of deploying the widget for use in `Mac OS X v10.4.3` or later or for use in all versions of `Mac OS X v10.4`. In most cases, deploying a widget for use in `Mac OS X v10.4.3` or later is advisable, because it results in a smaller widget.

If you want to share your widget by, for example, sending it in an email, click `Save to Disk` in the share pane and choose a convenient save location. If you want to make sure the widget project is automatically saved to disk before deploying, select the "Save project to disk before deploying" checkbox. Note that this checkbox is available after you've saved the widget project for the first time.

If the widget is for your use only, choose `File > Deploy Widget` to install it on your computer.

Deploying a Web Application

When you're ready to deploy your web application, you choose where you want Dashcode to place your application's code and resource files and how they should be saved. These files include the HTML, CSS, and JavaScript files, image files, and your application's Web Clip icon. (Note that Dashcode removes some Dashcode-specific content from the HTML file during deployment.)

By default Dashcode deploys these files to a folder in your `~/Library/Sites` directory, using the `localhost` account. You can modify this behavior by making choices in the `Share` pane (click `Share` in the navigator to open the `Share` pane).

Dashcode gives you three deployment options:

- **Deploy to localhost.** Dashcode places your web application files in a folder you specify in your `~/Library/Sites` directory.
- **Deploy to a remote server.** Dashcode uses WebDAV or MobileMe to upload your web application files to a server you specify.
- **Save to disk.** Dashcode saves your web application files in a folder whose name and location you supply.

If you want to save a copy of your web application files in a folder on disk, click **Save to Disk** at the bottom of the **Share** pane and supply a folder name and location. If you want to deploy the application to a local or remote server, you can customize the deployment behavior in the **Destination** section of the **Share** pane:


- Use the **Path** field to supply a name for the folder (otherwise, Dashcode creates a folder called **Untitled**).
- Use the **Account** pop-up menu if you want to select an account other than the default localhost account.
If you want to add a new account, select **Add Account** in the pop-up menu to open Dashcode's **Accounts** preferences and set up a new account.

In the **Options** section of the **Share** pane, you can determine save behavior and deployment notification:

- Select the **Save behavior** checkbox if you want your web application project to be saved before every deployment (this checkbox is available after you've saved the project for the first time).
- Select the **Notification** checkbox if you want Dashcode to send an email message containing a link to your application every time you deploy it. Use the field below the **Notification** checkbox to supply the recipient's email address (or multiple email addresses, separated by commas).

After you finish making your selections, click **Deploy** at the bottom of the **Share** pane, or choose **File > Deploy Web Application**, to deploy your web application. To view your web application on an iPhone OS-based device, navigate to the location of the `index.html` file on the appropriate server.

Note: As mentioned above, Dashcode uses WebDAV or MobileMe to upload your application files to a remote server. It does not support SSH, FTP, or SFTP.

 **Warning:** Web browsers, including Safari on iPhone, implement a security model known as "same source." This means that webpages are not allowed to request information from Internet domains other than the one from which they came. If you have built a web application that fetches information from other websites to display, it will work when you test it in Dashcode's simulator application, but not when you install it in your web server and run it in iPhone or iPod touch.

There are ways to configure your web server and alter your web application to make it possible to fetch information from other websites, but these techniques are beyond the scope of this document. Contact the administrator of your web server for advice.

Advanced Topics for Widgets

In addition to the core tasks of designing, coding, and testing a widget, Dashcode offers additional, advanced features that help you develop a widget that fits your users' needs. This chapter covers localization techniques and how to include a plug-in written in Objective-C.

Localization

You supply localized strings for your widget in the Localization section of the widget attributes pane. To show the widget attributes pane, click **Widget Attributes** in the navigator.

To localize your widget for other languages, click the plus (+) button and choose the language. Next to the language name in the first table, you provide a localized name for your widget. This name is used for your widget in the Finder and in the widget bar in Dashboard.

The table on the right shows all the strings used in your widget. In this table, you double-click a term in the **Value** column and provide localized versions of your widget's strings. You add key-value combinations for localization by clicking the plus button, providing a unique key name, and entering the localized version of the string next to its key.

When you add a language for localization, a language project folder is added to the widget. In addition to the strings, you can place any localized resources, such as style sheets or images, in a language project folder. To show the language project folder, choose **View > Files** and look for the folders that end with the `.lproj` extension. For more information on widget localization, read *Localizing Widgets* in *Dashboard Programming Topics*.

To make it easy to test your widget's localization, choose **View > Customize Toolbar**. In the dialog, drag the **Language** pop-up menu to the project window's toolbar, then click **Done**. Now you can choose a language from the **Language** pop-up menu to see your widget in that language. If you click **Run** in the toolbar, your widget runs in the selected language.

Widget Plug-ins

Dashcode offers a place to associate a widget plug-in with your widget. If you've built a custom widget plug-in with Xcode and want to include it with your widget, select **Widget Attributes** from the navigator and, in the **Plugin Name** section, click the **Choose** button and select a built widget plug-in.

For more information about widget plug-ins, read *Creating a Widget Plug-in* in *Dashboard Programming Topics*.

Dashcode Templates

Apple provides a number of templates with Dashcode. Read this appendix to learn more about the features of each of the Dashcode templates.

Widget Templates

The Custom Widget Template

The Custom template for widgets creates a blank widget that contains the basic files and images needed to make a widget. A widget made from this template contains no other preconfigured abilities.

The Countdown Template

The Countdown template creates a widget with a preconfigured digital-style countdown timer. To customize this template, add a picture or other artwork and set the target event for the countdown. To set the target event for the countdown, click Widget Attributes in the navigator and modify the options available under Properties.

Options that you can set include:

Target Kind

Set a specific date and time or a shared iCal calendar as the target for your widget. If you associate your widget with a shared iCal calendar, it counts down towards the next event on the calendar. After that event, the widget counts down to the next event on the calendar, and so forth. For more information on iCal and calendar publishing, read [Mac 101, Lesson 10: iCal](#).

When Reached

When the target event is reached, your widget can stop counting or start counting upwards from the event. Also, if you choose Do Action and provide a handler, it can trigger a JavaScript function, allowing your widget to respond to the event programmatically.

Display

Select whether the separator colons between the units of time flash and whether leading zero characters for each single-digit unit of time are shown.

The Maps Template

The Maps template creates a widget that displays locations and location-specific information on a map. To customize this template, change its appearance on the canvas and provide the URL of a map you publish. Also, you can change how your widget displays location information.

The Maps template works with KML files and GeoRSS feeds. To set the target map for your widget, click Widget Attributes in the navigator and modify the options available under Properties.

Options you can set include:

Maps API Key

Supply your Maps API key here. For more information on signing up for the Maps API, see [Google Maps API](#) on the Google developer website.

Initial Address

Supply the starting address you want to display in your widget here.

Mashup URL

Paste the URL for your map feed here. For more information on publishing personalized maps, see [How to Create a Map](#) on the Google developer website.

Note: You may find that you get the best results when you limit the number of points in your KML file to about 60 and limit the file's uncompressed size to about 1 MB.

The RSS Widget Template

The RSS template creates a widget that displays headlines from an RSS source. It is suited for RSS feeds that update very frequently because it shows a number of headlines at once. To customize this template, change its appearance on the canvas and provide an RSS feed URL for the widget to display. To set the target RSS feed for your widget, click Widget Attributes in the navigator and modify the options available under Properties.

Options that you can set include:

Feed URL

Paste the URL for an RSS feed here.

Show Articles

Adjust how many articles your widget shows and within what period of time the articles originate.

Display

Select whether date and time and whether a new content badge is shown with articles.

The Podcast Template

The Podcast template creates a widget that displays and plays episodes from a podcast. To customize this template, change its appearance on the canvas and provide a podcast URL for the widget to play back. To set the target podcast feed for your widget, click Widget Attributes in the navigator and modify the options available under Properties.

Options that you can set include:

Podcast URL

Paste the URL for a podcast here.

Check for updates

Set how often the widget should check for new episodes.

The Photocast Template

The Photocast template creates a widget that displays a slideshow of pictures obtained from an iPhoto Photocast. To customize this template, change its appearance on the canvas and provide a Photocast URL for the widget to display. To set the target Photocast for your widget, click Widget Attributes in the navigator and modify the options available under Properties.

Options that you can set include:

Photocast URL

Paste the URL for a Photocast here.

Change Picture

Adjust how often pictures should change.

Transition and Direction

Set the transition used when pictures change and, if available, the direction the transition should occur.

Show Title

Set the conditions under which the title of the Photocast should display.

The Quartz Composer Template

The Quartz Composer template creates a widget that displays a Quartz Composer composition. Quartz Composer compositions can be used in widgets to process and display graphical data. To customize this template, show the Files list and open `Default.qtz` in Quartz Composer. Quartz Composer is installed as part of the Developer Tools for Mac OS X v10.5.

For more information on using Quartz Composer, read *Quartz Composer Programming Guide*. For more information on the Quartz Composer WebKit plug-in (the plug-in that powers the Quartz Composer template) read *Quartz Composer WebKit Plug-in JavaScript Reference*.

Note: Widgets created using the Quartz Composer template are compatible with Mac OS X v10.4.7 and later.

The Video Podcast Template

The Video Podcast template creates a widget that displays and plays episodes from a video podcast. To customize this template, change its appearance on the canvas and provide a video podcast URL for the widget to play back. To set the target video podcast feed for your widget, click Widget Attributes in the navigator and modify the options available under Properties.

Options that you can set include:

Podcast URL

Paste the URL for a video podcast here.

Check for updates

Set how often the widget should check for new episodes.

The Gauge Template

The Gauge template creates a widget with gauges and indicators useful for monitoring activities. To change the sources of data for the gauges and indicators, choose `CommandMonitor.js` from the Files list and write code to interpret your source data and feed results to the gauges and indicators.

The Daily Feed Template

The Daily Feed template creates a widget that displays individual articles or images from an RSS source. It is suited for RSS feeds that don't update frequently since it shows one article at a time. To customize this template, change its appearance on the canvas and provide an RSS feed URL for the widget to display. To set the target RSS feed for your widget, click **Widget Attributes** in the navigator and modify the options available under **Properties**.

Options that you can set include:

Feed URL

Paste the URL for an RSS feed here.

Feed Type

Specify what kind of information the feed provides. If it provides HTML content, choose **HTML**. If your feed provides an image (such as a comic strip or daily photo), choose **Image**.

Web Application Templates

The Custom Web Application Template

The Custom template for web applications creates a blank web application that contains the basic files and images to make a web application. A web application developed with the Custom template is suitable to run in Safari on iPhone and iPod touch. A web application made from this template contains no other preconfigured abilities.

The Browser Template

The Browser template creates a web application that supports navigation through multiple levels of content. A web application developed with the Browser template is well-suited to displaying hierarchical information through which users can drill down.

The default web application provided by the Browser template contains a list view for the top level of the content and a detail view for the second level of the content. Selecting an item in the top-level list automatically reveals the detail view associated with that item. You can add additional list levels and you can add content to the detail views.

To customize this template, you can adjust how users experience the application in the device, and you can use different parts, such as the column layout, in the detail-level view. To set how users view your web application in iPhone and iPod touch, click **Application Attributes** and modify the options available under **Viewport**.

Options that you can set include:

When device orientation changes

Set whether your web application should change its width or zoom.

Page Zooming

Set whether users should be able to zoom your web application and, if so, the maximum zoom factor.

The Podcast Web Application Template

The Podcast template creates a web application that displays and plays episodes of a podcast that you publish. The template is set up to display a list of episodes that are played when users select them.

To customize this template, provide the URL of a podcast that you publish. To do this, click Application Attributes in the navigator and modify the option under Properties. (You can also set values that define how users can view your web application, as described in [“The Browser Template”](#) (page 56).)

The option you can set is:

Podcast URL

Paste the URL for the podcast you publish here.

The RSS Web Application Template

The RSS template creates a web application that displays headlines and articles from an RSS source. A web application developed with this template can handle RSS feeds that update very frequently because the template allows you to show numerous headlines at once.

To customize this template, provide an RSS feed URL that you publish. You can also adjust how many articles are displayed and for how long. To customize these attributes, click Application Attributes in the navigator and modify the options under Properties. (You can also set values that define how users can view your web application, as described in [“The Browser Template”](#) (page 56).)

Options that you can set include:

Feed URL

Paste the URL for the feed you publish here.

Show Articles

Adjust how many articles to display in your web application, within what time period the articles originate, and how many “top stories” to display.

The Utility Web Application Template

The Utility template creates a web application that displays primary information on the front side and various view or configuration options on the reverse side. A web application developed with this template is well-suited to display small amounts of targeted information in a format reminiscent of Dashboard widgets.

To customize this template, place your content in the front view and the view or configuration options you want to offer in the settings view. Be sure to leave the info button in place (the template provides this button in the lower-right corner of the front view), because users know they can tap this control to see the back view of the application. Click Application Attributes in the navigator to set values that define how users can view your web application, as described in [“The Browser Template”](#) (page 56).

In its `main.js` JavaScript file, the Utility template also demonstrates how to use client-side database storage to store content that can be used whether or not the application is online. The template includes code that sets up a simple key-value table that holds values for the message displayed on the front and its font, size, and color.

Dashcode Parts

Dashcode includes a number of unique elements that you use on your widget's or web application's interface to display information. Some of these elements, called Dashcode parts, can be modified programmatically. For those that can be modified programmatically, this appendix describes some of the methods you can use to do so. To look at the code that implements these parts, choose the Files view in the navigator and select the appropriate file.

Some parts listed in this appendix are specifically designed to work with web applications, not widgets, and are described as such.

If a part is not listed in this appendix, there is no Dashcode-specific programming interface needed to use it. You can change its attributes using the Attributes inspector and assign it event handlers using the Behaviors inspector. For more information about the inspector window, read ["Changing an Element's Properties"](#) (page 32).

Activity Indicator

The Activity Indicator part is designed for use in a web application. An activity indicator shows the user that a task or process is progressing, but does not indicate when it will finish.

The Activity Indicator part includes JavaScript code that starts and stops the spinning action. You can use the `startAnimation` and `stopAnimation` methods to coordinate the spinning of an activity indicator with the progress of a task or process.

Back Button

The Back button part is designed for use in a web application. In a web application that displays multiple levels of information, a back button gives users a convenient way to retrace their steps within the application. This allows an application to present its own navigational hierarchy within a single URL.

The Back button part is automatically included in the code for a project based on the Browser template. It is set up to appear on all levels except the first.

Browser

The Browser part is designed for use in a web application. A browser provides an area for grouping elements and browsing back and forth. When you select the Browser web application template, a Browser part is included automatically.

The Browser part includes JavaScript code to perform some standard actions. In particular, a browser supplies a method you can use to reveal the next level in a hierarchy of information. Typically, you call the `goForward` method in the controller code for a list part, assigning it to handle the `onClick` event associated with the forward arrow. The Browser part handles the analogous “go back” functionality automatically.

Canvas

The Canvas part is a custom drawing region you can add to your widget. Using the Canvas discusses how to draw on a canvas using JavaScript.

Call Button

The Call button part is designed for use in a web application. A call button gives users a convenient way to initiate a phone call. In the Attributes inspector, you can specify a default phone number to dial when users tap the button.

The Call button part includes JavaScript code to perform some standard actions, such as methods to get and set the phone number. If the button is enabled, the default action is to initiate a phone call to the currently set phone number when the user taps it. You can use the `setPhoneNumber` method to allow users to change the phone number dynamically.

Column Layout

The Column Layout part is an area you can use for laying out content side by side. This part is especially useful for web applications because each column can be set to position its content absolutely or relatively.

If, for example, you wanted to provide one column to display fixed-size images and another column to display variably sized descriptions about the images, you could specify the first column to use absolute positioning and the second column to use relative positioning. The Column Layout part handles much of this for you.

Edge-to-Edge List

The Edge-to-Edge List part provides a list format in which each row stretches from one side of its container to the other. An edge-to-edge list contains a customizable row template that is used to create new rows in the list.

You can specify static data to display in each row, or you can customize the list’s controller code to provide data dynamically at runtime. You can set this in the Attributes inspector.

An edge-to-edge list that receives data dynamically needs a controller object (a static edge-to-edge list does not need a controller object). The list controller object must implement two required data source methods: `numberOfRows` and `prepareRow`. The `numberOfRows` callback method returns the total number of rows in the list. The `prepareRow` method is called once for each row as it is being prepared and it uses the values it is passed to populate the row. Specifically, the `prepareRow` method requires the following three arguments:

- `rowElement`. This is the enclosing element for the entire row. You can use this element to install click handlers for each row in the list.
- `rowIndex`. This is the zero-based index of the row. You can use this value to find the appropriate row data in application-specific data structures.
- `templateElements`. This is a dictionary that contains an entry for each element in the template row that has an ID. Each key in the dictionary is an ID and the associated value is the element within `rowElement` that corresponds to the original template elements, such as `rowTitle`. You can use the values associated with these elements to fill in the appropriate places in the row.

An edge-to-edge list also contains the public `rows` property, which gives you access to each row element in the list. If the list is dynamic, you can use the `rows` property to view currently filled rows in your `prepareRow` callback method while the list is filling, but you won't be able to see all the rows until the list is completely filled. For both static and dynamic lists, the `rows` property is most useful for getting access to specific rows after the list is filled.

If your edge-to-edge list receives data dynamically, be sure to call the `reloadData` function to force the list to reload and display the updated content. (Note that `reloadData` is called automatically when the list part is initialized.) To do this, add this line of code to your list controller:

```
document.getElementById("myList").object.reloadData();
```

Forward Button

The Forward button part is designed for use in web applications. A forward button gives users a convenient way to move forward through the multiple levels of a web application.

Gauge

The Gauge part is a dial with a pointer that indicates a value in a range of values. You can edit a gauge's appearance and range of values using the Attributes inspector. If you're using a gauge as a control, select the Track Mouse Down option in the Attributes inspector and supply your own handler function for the `onchange` event in the Behaviors inspector. Your handler is called whenever the gauge's value changes.

If you're using a gauge to graphically represent data, you need to update its value using JavaScript. Use the `setValue` method to update a gauge's value, as shown here:

```
document.getElementById("gauge").object.setValue(50);
```

Note that the value provided to `setValue` should lie within the range specified in the Attributes inspector for the gauge.

Indicator

The Indicator part is a light that changes color at different values. You can edit an indicator's appearance and range of values using the Attributes inspector.

When using an indicator, you need to update its value using JavaScript, as shown here:

```
document.getElementById("indicator").object.setValue(10);
```

Note that the value provided to `setValue` should lie within the range specified in the Attributes inspector for the indicator.

Level Indicators

The Horizontal and Vertical Level Indicator parts are linear indicators that show a value in a range of values. You can edit a level indicator's appearance and range of values using the Attributes inspector. If you're using a level indicator as a control, select the Track Mouse Down option in the Attributes inspector and supply your own handler function for the `onchange` event in the Behaviors inspector. Your handler is called whenever the level indicator's value changes.

If you're using a level indicator to graphically represent data, you need to update its value using JavaScript. Use the `setValue` method to update a level indicator's value, as shown here:

```
document.getElementById("horizontalLevelIndicator").object.setValue(10)
```

Note that the value provided to `setValue` should lie within the range specified in the Attributes inspector for the level indicator.

Mail Button

The Mail button part is designed for use in a web application. A mail button gives users a convenient way to initiate an email message. In the Attributes inspector, you can specify a default recipient (email address) and subject to use when users tap the button.

The Mail button part includes JavaScript code to perform some standard actions, such as methods to get and set the email address and to get and set the subject. If the button is enabled, the default action is to open a message composing view, with the To and Subject fields filled in with the currently set address and subject. You can use the `setEmailAddress` and `setSubject` methods to allow users to change the email address and subject dynamically.

Map Button

The Map button part is designed for use in a web application. A map button gives users a convenient way to view a map for a location. In the Attributes inspector, you can specify a default location to display when users tap the button.

The Map button part includes JavaScript code to perform some standard actions, such as methods to get and set the address to display. If the button is enabled, the default action is to display a map for the currently set location. You can use the `setAddress` method to allow users to change the map location dynamically.

Quartz Composer

The Quartz Composer part is an area that contains a Quartz Composer composition. To manipulate a composition while your widget runs, use the Quartz Composer WebKit plug-in's JavaScript API. You can learn more about the Quartz Composer WebKit plug-in's JavaScript API by reading *Quartz Composer WebKit Plug-in JavaScript Reference*.

QuickTime

The QuickTime part is an area used for playback of QuickTime media. Use the QuickTime JavaScript methods to control the playback of the movie or change its properties. To learn more about the QuickTime plug-in's JavaScript methods, read *JavaScript Scripting Guide for QuickTime*.

Rounded-Rectangle List

The Rounded-Rectangle List part provides a list in which a group of rows is inset from the edges of its container and is bordered by a rounded rectangle. A rounded-rectangle list part contains a customizable row template you can use to create new rows in the list.

You can specify static data to display in each row, or you can customize the list's controller code to provide data dynamically at runtime. You can set this in the Attributes inspector.

A rounded-rectangle list that receives data dynamically needs a controller object (a static rounded-rectangle list does not need a controller object). The list controller object must implement two required data source methods: `numberOfRows` and `prepareRow`. The `numberOfRows` callback method returns the total number of rows in the list. The `prepareRow` method is called once for each row as it is being prepared and it uses the values it is passed to populate the row. These values are passed to the `prepareRow` method in the following three arguments:

- `rowElement`. This is the enclosing element for the entire row. You can use this element to install click handlers for each row in the list.
- `rowIndex`. This is the zero-based index of the row. You can use this value to find the appropriate row data in application-specific data structures.
- `templateElements`. This is a dictionary that contains an entry for each element in the template row that has an ID. Each key in the dictionary is an ID and the associated value is the element within `rowElement` that corresponds to the original template elements, such as `rowTitle`. You can use the values associated with these elements to fill in the appropriate places in the row.

A rounded-rectangle list also contains the public `rows` property, which gives you access to each row element in the list. If the list is dynamic, you can use the `rows` property to view currently filled rows in your `prepareRow` callback method while the list is filling, but you won't be able to see all the rows until the list is completely filled. For both static and dynamic lists, the `rows` property is most useful for getting access to specific rows after the list is filled.

If your rounded-rectangle list receives data dynamically, be sure to call the `reloadData` function to force the list to reload and display the updated content. (Note that `reloadData` is called automatically when the list part is initialized.) To do this, add this line of code to your list controller:

```
document.getElementById("myList").object.reloadData();
```

Scroll Area

The Scroll Area part is an area wrapped with scroll bars, meant for showing content larger than a widget's or web application's interface. You can customize a scroll area's appearance using the Attributes inspector. Options include whether the scroll bar automatically hides when its contents fit in its bounds and the dimensions of its bounds and margins.

To change the content of a scroll area, use the `content` property from the scroll area's object, as shown here:

```
var content = document.getElementById("scrollArea").object.content;
content.innerHTML = someText;
```

Once you obtain the scroll area's `content` property, you have access to the `<div>` element that is inside the scroll area. From there, you can use the `innerHTML` or `innerHTML` properties to change the scroll area's contents.

Stack Layout

The Stack Layout part is designed for use in web applications. A stack layout is an area that contains views that can be swapped.

You can use the Attributes inspector to add and remove views in the stack layout, and you can specify transition styles to be used while swapping. For example, you can change the transition between views from a push (in which the new view pushes the old view from the side) to a fade (in which the new view fades in over the top of the old view).

Stack Layout Methods

A stack layout part includes a number of methods you can use to access its subviews and set transitions between views. In particular, you can use:

- `getCurrentView`. This method returns the currently active view.
- `getAllViews`. This method returns a JavaScript array of views in the stack layout. You might want to use the index of the array to get the "next" view.

- `setCurrentView`. This method takes the following parameters:
 - `newView`. The new view to display.
 - `isReverse`. A Boolean value that specifies whether the transition should be performed in reverse (for example, the push transition defines a reverse transition).
 - `makeTopVisible`. A Boolean value that specifies whether the new view should be scrolled to the top.

The `setCurrentView` method sets the current view to the passed-in view and sets the transition and scroll position appropriately.

- `setCurrentViewWithTransition`. This method is similar to `setCurrentView` except that it includes a transition parameter after the `newView` parameter that allows you to specify a transition. (See “Stack Layout Transitions” for more information about transition objects.)

Stack Layout Transitions

The transition object handles transitions between views in a stack layout container. By default the transition object handles a few transition types and a number of transition properties. Transition objects are created with the `Transition` constructor function, as shown here:

```
Transition (type, duration, timing)
```

The parameters of the `Transition` function specify the type of transition, the length of time the transition takes, and the acceleration curve of the transition. Note that you can also set these properties in the inspector after a transition object has been created.

The available transition types are:

- `Transition.NONE_TYPE`. The new view simply appears in place of the old view.
- `Transition.PUSH_TYPE`. A two-dimensional transition in which the new view pushes the old view off one edge of the screen.
- `Transition.DISSOLVE_TYPE`. A two-dimensional transition in which the old view dissolves into the new view.
- `Transition.SLIDE_TYPE`. A two-dimensional transition in which the new view slides in over the old view from one edge of the screen.
- `Transition.FADE_TYPE`. A two-dimensional transition in which the new view fades in over the old view.
- `Transition.FLIP_TYPE`. A three-dimensional transition in which the old view flips over to reveal the new view.
- `Transition.CUBE_TYPE`. A three-dimensional transition in which the old view and new view appear as if they are adjacent faces of a cube, which rotates to reveal the new view.
- `Transition.SWAP_TYPE`. A three-dimensional transition in which both views move away from each other horizontally and then swap positions vertically, leaving the new view on top.
- `Transition.REVOLVE_TYPE`. A three-dimensional transition in which the old view and the new view share an axis on one edge, about which the old view revolves out as the new view revolves in.

The `duration` parameter should contain the number of seconds the transition should take, from start to finish. Finally, for the `timing` parameter, you can specify a timing function that controls the speed and acceleration of the transition (the timing functions are defined in the WebKit CSS animation specification). The available transition timing functions are:

- `Transition.EASE_TIMING`
- `Transition.LINEAR_TIMING`
- `Transition.EASE_IN_TIMING`
- `Transition.EASE_OUT_TIMING`
- `Transition.EASE_IN_OUT_TIMING`

The timing functions determine the acceleration curve of the transition. A linear curve means constant speed (that is, no acceleration), and “ease” means gaining or losing speed gradually. For example, the `Transition.EASE_IN_TIMING` function means that the transition starts slow and gradually gets faster, whereas the `Transition.EASE_OUT_TIMING` function means that the transition gradually slows down near the end.

You can also specify a direction for the `Transition.PUSH_TYPE` and `Transition.SLIDE_TYPE` transitions. The available directions are:

- `Transition.RIGHT_TO_LEFT_DIRECTION`
- `Transition.LEFT_TO_RIGHT_DIRECTION`
- `Transition.TOP_TO_BOTTOM_DIRECTION`
- `Transition.BOTTOM_TO_TOP_DIRECTION`

Finally, you specify a direction for the four three-dimensional transitions. The available directions for these transitions are:

- `Transition.RIGHT_TO_LEFT_DIRECTION`
- `Transition.LEFT_TO_RIGHT_DIRECTION`

The transition object includes the `perform` method that performs the transition. You pass in the following parameters to the `perform` method:

- `newView`. The view that will be visible after the transition.
- `oldView`. The view that is visible now (before the transition).
- `isReverse`. A Boolean flag that specifies whether the transition should be performed in reverse. (The push transition, for example, can be performed in reverse; the cross-fade transition is the same either way.)

The `perform` method ensures that the passed-in views share a common parent container element, which is important because the transition is constrained by the dimensions of the parent container. In particular, the container must constrain the transitions for overflow content.

Note that you can use transition objects to pass in for the transition parameter of the stack layout `setCurrentViewWithTransition` method. For more information about the stack layout methods, see [“Stack Layout Methods”](#) (page 64).

Document Revision History

This table describes the changes to *Dashcode User Guide*.

Date	Notes
2009-03-04	Made minor corrections.
2009-01-06	Corrected the description of web application deployment.
2008-09-09	Updated for Dashcode 2.0.1.
2008-07-11	Added a new tutorial for web applications and descriptions for new templates and parts introduced in Dashcode 2.0.
2007-08-07	New document that describes how to create Dashboard widgets with Dashcode.

REVISION HISTORY

Document Revision History