"The Missing Manual series is simply the most intelligent and usable series of guidebooks..."
—KEVIN KELLY, CO-FOUNDER OF *WIRED*

Adobe Edge Preview 5

the missing manual®

The book that should have been in the box®

Includes free ebook and updates

O'REILLY®

Chris Grover

HTML5 / WEB DESIGN

Answers found here!

Want to create animated graphics for iPhone, iPad, and the Web, using familiar Adobe features? You've come to the right guide. *Adobe Edge Preview 5: The Missing Manual* shows you how to build HTML5 and JavaScript graphics with Adobe multimedia tools. No programming experience? No problem. Adobe Edge writes all the code for you. With this book, you'll be designing great-looking web apps in no time.

the missing manual®

The book that should have been in the box®

The important stuff you need to know

- **Get to know the workspace.** Learn how Adobe Edge Preview 5 performs its magic.
- **Create and import graphics.** Make drawings with Edge's tools, or use art you designed in other programs.
- **Work with text.** Build menus, label buttons, provide instructions, and perform other tasks.
- **Jump into animation.** Master Edge's elements, properties, and timeline panels.
- **Make it interactive.** Use triggers and actions to give users control over their web experience.
- **Peek behind the curtain.** Understand how HTML and CSS documents work.
- **Dig into JavaScript.** Customize your projects by tweaking your code.

Bestselling author **Chris Grover** has more than 25 years experience in graphic design and electronic media. He excels in making complex technology fun and easy to learn. In this book, Chris applies the winning formula he used in *Flash CS5.5: The Missing Manual* and *Google SketchUp: The Missing Manual.*

US $24.99 CAN $26.99
ISBN: 978-1-449-33030-9

O'REILLY®

missingmanuals.com
twitter: @missingmanuals
facebook.com/MissingManuals

Adobe Edge Preview 5

the missing manual®

The book that should have been in the box®

Adobe Edge Preview 5

the missing manual®

The book that should have been in the box®

Chris Grover

O'REILLY®

Beijing | Cambridge | Farnham | Köln | Sebastopol | Tokyo

Adobe Edge Preview 5: The Missing Manual

by Chris Grover

Published by O'Reilly Media, Inc.,
1005 Gravenstein Highway North, Sebastopol, CA 95472.

May 2011:	First Edition.
December 2011:	Second Edition.
April 2012:	Third Edition.

Revision History for the 3rd Edition:

2011-05-26	First release
2011-08-26	Second release (ebook only)
2012-04-27	Third release

See *http://oreilly.com/catalog/errata.csp?isbn=9781449330309* for release details.

ISBN-13: 978-1-449-33030-9

Contents

Introduction

It may be hard to imagine, but once upon a time, pages on the World Wide Web didn't have pictures, let alone animations, videos, and interactive graphics. All these elements were added through trial, error, debate, and debunk. Changes came when brave souls (like you) forged ahead and made things work with the tools at hand. If a commercial product worked well and was widely adopted, it became the de facto standard. Adobe's PDF (portable document files) and Flash animation player are well-known examples. However, there's always been a problem with proprietary and patent-encumbered technologies on the Internet. They're like a toll road in the center of a major city. On the other hand, authorities and standards-writing groups have been known to create "standards" that few browser and web developers follow. Strictly structured XHTML pages fall into this category. The solution is to create standards for the Internet that are practical, usable, and don't stifle creativity. Of course, that's easier said than done.

TIP As Adobe Edge continues to grow and change, we'll be updating this book periodically. Your purchase of this book includes free updates to the ebook edition. To get your free ebook, go to *http://shop.oreilly.com/product/0636920025658.do*. Next to the Ebook buying option, click "Add to Cart." Enter AE5MM in the Discount Code box; click Submit; click Checkout; and then follow the onscreen instructions.

With HTML5, the standards-writing crowd (also known as the W3C) is working hard to give the Internet community a roadmap that takes into account where we've been and where we're heading. There are a number of exciting new features in HTML5, but perhaps most visible are the new ways to present and animate graphics. If you're thinking, "That sounds a lot like Adobe Flash," you're right. One shiny new feature of HTML5 provides a non-proprietary, standard way to change graphics, color, size, shape, and position over time. The technique uses newly defined HTML tags, the power of JavaScript, and its jQuery companion library. These open-source technologies are available to everyone, whether they're designing web pages or building the next great web browser.

Why Use Adobe Edge?

If you need a compelling reason to learn yet another animation technology, here are three good ones: iPhone, iPod, and iPad. In fact, if you're a Flash designer or developer, you're probably already dialed in to the famous debate between Apple and Adobe regarding Flash. As a web designer and developer, more important than the debate is the fact that Flash content on web pages can't be viewed by the most popular mobile devices on the planet. However, if you use HTML5 and JavaScript, you can capture that Apple audience and more.

So why should you use an Adobe product to create HTML5 web content? It's an understatement to say that most graphic artists view the world differently from computer programmers. If you're an artist, you may not be entirely comfortable describing each circle, gradient, and line in your artwork by typing out JavaScript code, even though it's theoretically possible. You're probably more inclined to use a tool that reminds you of Adobe Illustrator, Photoshop, or Flash. That's exactly where Edge fits in. Edge has a Timeline like the ones in Flash and After Effects. The Elements and Properties panels will remind you of your favorite drawing and photo tools. If you use Edge to develop HTML5 graphics, then you can concentrate on creating and fine-tuning your artwork. Edge will generate the HTML5 and JavaScript/jQuery code that's needed for your web pages.

Where to Find Adobe Edge

If you don't already have Adobe Edge on your computer, you may be wondering where to get it. At this writing, you couldn't buy Edge from Amazon. That's because at the moment, it's free "preview" software from Adobe Labs. To download the program, go to *http://labs.adobe.com/technologies/edge/*. Click the Download Now button. If you don't have an Adobe account, you'll need to create one before you can download the software. You know the drill: name, email, password. Both the account and Edge are free. Adobe is not saying whether Edge, or a similar product, will continue to be free in the future. If you want more details on how to install Edge on your computer, check out Appendix A.

About This Book

Despite the many improvements in software over the years, one feature has grown consistently worse: documentation. With the purchase of most software programs these days, you don't get a single page of printed instructions. To learn about the hundreds of features in a program, you're expected to use online electronic help, but with a Preview product like Adobe Edge, the help files are missing or incomplete.

But even if you're comfortable reading a help screen in one window as you try to work in another, something is still missing. At times, the terse electronic help screens assume that you already understand the discussion at hand and hurriedly skip over important topics that require an in-depth presentation. In addition, you don't always get an objective evaluation of the program's features. (Engineers often add technically sophisticated features to a program because they *can,* not because you need them.) You shouldn't have to waste your time learning features that don't help you get your work done.

The purpose of this book, then, is to serve as the manual that should have been in the box. In this book's pages, you'll find step-by-step instructions for using every Edge feature, including those you may not even have quite understood, let alone mastered, such as moving the HTML5 and JavaScript code into your web pages or making changes to existing pages using Edge. In addition, you'll find clear evaluations of each feature that help you determine which ones are useful to you, as well as how and when to use them.

NOTE This book periodically recommends *other* books, covering topics that are too specialized or tangential for a manual about Edge. Careful readers may notice that not every one of these titles is published by Missing Manual-parent O'Reilly Media. While we're happy to mention other Missing Manuals and books in the O'Reilly family, if there's a great book out there that doesn't happen to be published by O'Reilly, we'll still let you know about it.

Adobe Edge Preview 5: The Missing Manual is designed to accommodate readers at every technical level. The primary discussions are written for advanced-beginner or intermediate computer users. But if you're a first-timer, special sidebar articles called Up to Speed provide the introductory information you need to understand the topic at hand. If you're an advanced user, on the other hand, keep your eye out for similar shaded boxes called Power Users' Clinic. They offer more technical tips, tricks, and shortcuts for the experienced computer fan.

Macintosh and Windows

Edge works almost precisely the same in its Macintosh and Windows versions. Every button in every dialog box is exactly the same; the software response to every command is identical. In this book, the illustrations have been given even-handed treatment, rotating between the two operating systems where Edge is at home (Windows 7 and Mac OS X).

One of the biggest differences between the Mac and Windows versions is the keystrokes, because the Ctrl key in Windows is the equivalent of the ⌘ key on the Mac.

Whenever this book refers to a key combination, you'll see the Windows keystroke listed first (with + symbols, as is customary in Windows documentation); the Macintosh keystroke follows in parentheses (with - symbols, in time-honored Mac fashion). In other words, you might read, "The keyboard shortcut for saving a file is Ctrl+S (⌘-S)."

About the Outline

Adobe Edge Preview 5: The Missing Manual is divided into three parts, each containing several chapters:

- **Part One: Working with the Stage** starts off with an introduction to the Edge workspace. You'll learn some more details about how Edge performs its magic by creating HTML, JavaScript, and CSS code. Then you'll roll up your sleeves and create graphics within Edge and import artwork from other programs. Along the way, you'll begin to work with Edge's Timeline and Properties panel to make things move. Chapter 3 is devoted to working with text and you'll see how easy it is to make text elements change size, shape, and color.
- **Part Two: Animation with Edge** is all about animating the elements on the stage. You'll learn advanced techniques for working efficiently in Edge. Animation is time-consuming work, but you can save lots of time by reusing and recycling your previous work. You'll learn to manage and edit the Timeline and the transitions you create. The last chapter in this section is devoted to triggers and actions. You use these tools to automate your animation and give interactive control to your web pages.
- **Part Three: Edge with HTML5 and JavaScript** gets into the nitty-gritty details of working with code. This book doesn't try to be an advanced JavaScript programmer's manual. Instead, you'll learn how to selectively tweak bits of code to make some animation magic.

Appendix A explains how to install Edge on both Windows and Mac computers. You'll also find tips on where to look for discussions and additional Edge resources.

The Very Basics

You'll find very little jargon or nerd terminology in this book. You will, however, come across a few terms and concepts that you'll encounter frequently in your computing life:

To use this book (and indeed to use Adobe Edge), you need to know a few basics. This book assumes that you're familiar with a few terms and concepts:

- **Clicking.** This book includes instructions that require you to use your computer's mouse or trackpad. To *click* means to point your cursor (the arrow pointer) at something on the screen and then—without moving the cursor at all—press and release the left button on the mouse (or laptop trackpad). To *right-click* means the same thing, but pressing the *right* mouse button instead. (Usually, clicking selects an onscreen element or presses an onscreen button, whereas right-clicking typically reveals a *shortcut menu*, which lists some common tasks specific to whatever you're right-clicking.) To *double-click*, of course, means to click twice in rapid succession, again without moving the pointer at all. And to *drag* means to move the cursor while holding down the (left) mouse button the entire time. To *right-drag* means to do the same thing but holding down the right mouse button.

  When you're told to *Shift-click* something, you click while pressing the Shift key. Related procedures, like *Ctrl-clicking*, work the same way—just click while pressing the corresponding key.

- **Menus.** The *menus* are the words at the top of your screen: File, Edit, and so on. Click one to make a list of commands appear, as though they're written on a window shade you've just pulled down. Some people click to open a menu and then release the mouse button; after reading the menu command choices, they click the command they want. Other people like to press the mouse button continuously as they click the menu title and drag down the list to the desired command; only then do they release the mouse button. Both methods work, so use whichever one you prefer.

- **Keyboard shortcuts.** Nothing is faster than keeping your fingers on your keyboard to enter data, choose names, trigger commands, and so on—without losing time by grabbing the mouse, carefully positioning it, and then choosing a command or list entry. That's why many people prefer to trigger commands by pressing combinations of keys on the keyboard. For example, in most word processors, you can press Ctrl+B to produce a boldface word. In this book, when you read an instruction like "Press Ctrl+L to insert a label," start by pressing the Ctrl key; while it's down, type the letter L; and then release both keys.

About→These→Arrows

Throughout this book, and throughout the Missing Manual series, you'll find sentences like this one: "Open the System Folder→Preferences→Remote Access folder." That's shorthand for a much longer instruction that directs you to open three nested folders in sequence, like this: "On your hard drive, you'll find a folder called System Folder. Open that. Inside the System Folder window is a folder called Preferences; double-click it to open it. Inside that folder is yet another one called Remote Access. Double-click to open it, too." Similarly, this kind of arrow shorthand helps to simplify the business of choosing commands in menus, as shown in Figure I-1.

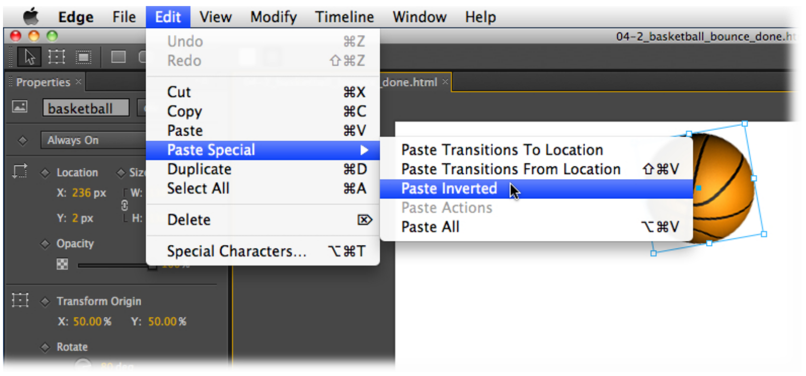


FIGURE I-1

When you read in a Missing Manual, "Choose Edit→Paste Special→Paste Inverted," that means: "Click the Edit menu to open it. Then click Paste Special in that menu; choose Paste Inverted in the resulting submenu."

About the Online Resources

As the owner of a Missing Manual, you've got more than just a book to read. Online, you'll find example files so you can get some hands-on experience, as well as tips, articles, and maybe even a video or two. You can also communicate with the Missing Manual team and tell us what you love (or hate) about the book. Head over to *www.missingmanuals.com*, or go directly to one of the following sections.

Missing CD

This book doesn't have a CD pasted inside the back cover, but you're not missing out on anything. Go to *www.missingmanuals.com/cds/edgepv5mm* to download all the examples and exercises that are covered in this book. You can download all the files in one big ZIP file or you can download the files chapter by chapter. Most examples are made up of several files, which might include a web page (.html), images (.jpg), JavaScript code (.js), and style sheets (.css), so it's important to keep the files and their folders together or the examples may not work. Example and exercise folders and files are numbered, so when you see *03_2-MyExample*, you'll know that this example is from Chapter 3 and it's the second folder for the chapter. For many of the exercises, there are completed examples that you can use to check your own work. A completed example includes the word *done* in the filename as in *03-3_MyExample_done*.

Finally, so you don't wear down your fingers typing long web addresses, the Missing CD page also offers a list of clickable links to the websites mentioned in this book.

Registration

If you register this book at oreilly.com, you'll be eligible for special offers—like discounts on future editions of *Adobe Edge Preview 5: The Missing Manual*. Registering takes only a few clicks. To get started, type *http://oreilly.com/register* into your browser to hop directly to the Registration page.

Feedback

Got questions? Need more information? Fancy yourself a book reviewer? On our Feedback page, you can get expert answers to questions that come to you while reading, share your thoughts on this Missing Manual, and find groups for folks who share your interest in web design and animation. To have your say, go to *www.missingmanuals.com/feedback*.

Errata

In an effort to keep this book as up-to-date and accurate as possible, each time we print more copies, we'll make any confirmed corrections you've suggested. We also note such changes on the book's website, so you can mark important corrections into your own copy of the book, if you like. Go to *http://tinyurl.com/edgepv5-mm* to report an error and view existing corrections.

Newsletter

Our free email newsletter keeps you up-to-date on what's happening in Missing Manual land. You can meet the authors and editors, see bonus video and book excerpts, and more. Go to *http://tinyurl.com/MMnewsletter* to sign up.

Safari® Books Online

Safari® Books Online is an on-demand digital library that lets you easily search over 7,500 technology and creative reference books and videos to find the answers you need quickly.

With a subscription, you can read any page and watch any video from our library online. Read books on your cellphone and mobile devices. Access new titles before they're available for print, and get exclusive access to manuscripts in development and post feedback for the authors. Copy and paste code samples, organize your favorites, download chapters, bookmark key sections, create notes, print out pages, and benefit from tons of other time-saving features.

O'Reilly Media has uploaded this book to the Safari Books Online service. To have full digital access to this book and others on similar topics from O'Reilly and other publishers, sign up for free at *http://my.safaribooksonline.com*.

PART

1

Working with the Stage

CHAPTER

1

Introducing Adobe Edge

Travelers on the World Wide Web expect strong graphics. They appreciate animation that contributes to the subject as long as it doesn't waste their time. Done well, animation draws attention to important details, shows how things work, and helps site navigation. But, graphics certainly weren't first and foremost when the Web was created. The language used to display web pages is called HTML—short for *HyperText Markup Language*. It has evolved and continues to adapt to current needs and new ideas. The latest step in that evolution is HTML5, which combined with other technologies like CSS3, JavaScript, and jQuery, presents the beautiful interactive pages you visit today.

Instead of creating graphics and visual effects manually by writing code, artists can use Adobe Edge—a tool that's a much better fit for designers. This chapter starts off by explaining how Edge works to write HTML code that a web browser can read. Then it offers a quick introduction to the main parts of the Edge workspace. Finally, you'll take Edge for a test drive, where you'll make an image move and create text that fades in and out. Your first hands-on experience will be quick and easy. Consider this first adventure an overview—the following chapters will reveal the details.

Creating and Saving Edge Projects

Edge's role in life is to help you make web pages that move. You design the graphics using familiar visual tools, while Edge writes the underlying code. It's as if you hired an HTML/CSS/JavaScript/jQuery coder for your design team. One good way to understand what goes on behind the scenes is to create and save an empty Edge project. Fire up Edge as you would any other application on your computer. That means the process is slightly different for Windows and Mac computers. If you plan on using Edge a lot (and why wouldn't you?) you can use any of the familiar tricks to create handy shortcuts. In Windows, you can pin an Edge shortcut to your Start menu or the taskbar. On a Mac, you can add Edge to the Dock.

Once Edge is running, you're greeted with a workspace with a number of panels and some Adobe "Get Started" advice. Don't worry about those details now; you'll explore them later in this chapter. Create a new folder on your desktop and call it *Edge Barebones*. Next, do the project creation two-step. Go to File→New and then File→Save As. When you Save As, you see that Edge uses a standard file/folder navigation window for your computer. Find the Barebones folder on your desktop and save your project with any name you want. Now, examine the contents of the folder. You'll find five files and a folder like the ones in *Figure 1-1*. If you've spent time developing web pages, you'll see some of the usual suspects and maybe a newcomer or two:

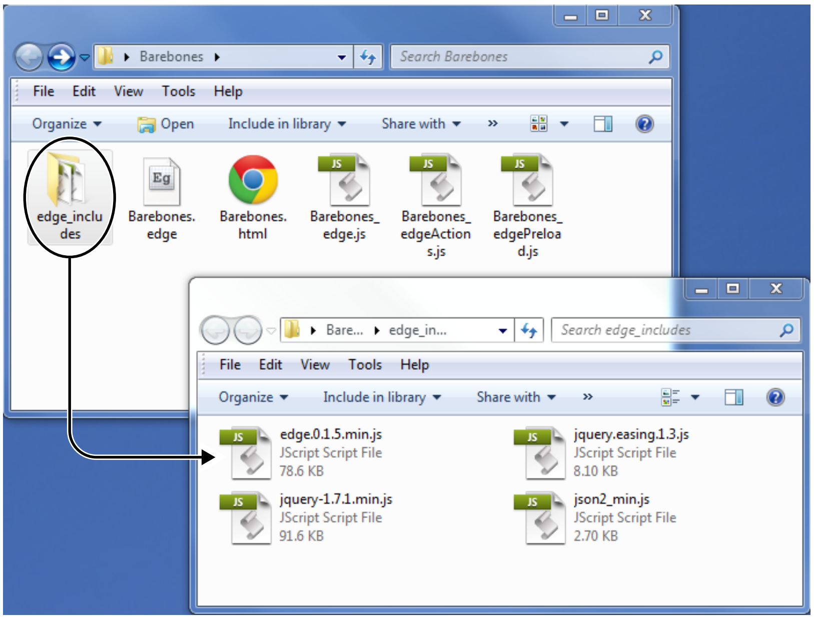
- The **.edge** file is used by Edge to keep track of your project.
- The **.html** file describes a web page using HTML code, like any of the gazillion web pages on the Internet.
- The **.js** files hold JavaScript code that's specific to your project. Right now your project is bare-bones, but the code defines the empty animation stage and performs other tasks that are necessary for all Edge projects.

Open the folder that's named *edge_includes,* and you'll find more JavaScript files. These are libraries of JavaScript code. One is specific to Edge; the others are standard JavaScript libraries. These libraries are referenced by the code in the HTML page that Edge created. They serve as the engine behind your Edge project. In short, they make things move.

Unlike a word processor or a spreadsheet, which create single files, Edge creates several files, and it needs those files to build the project and to display your masterfully designed page in a browser. If you delete or move one of these files, chances are you'll confuse Edge and anyone who views the web page. So one thing to learn from this bare-bones exercise is proper folder and file management:

- Create separate folders for each project you tackle, including the exercises in this book. (You may want to put them all in a main Edge project folder.)
- Don't delete, move, or rename the files and folders that Edge creates, until you fully understand their relationships.

NOTE Actually, all your projects could share the files in the *edge_includes* folder. For now, it's easiest to let Edge create new files for each project. They don't take up that much storage space on your computer.



edge.0.1.5.min.js
JScript Script File
78.6 KB

jquery.easing.1.3.js
JScript Script File
8.10 KB

jquery-1.7.1.min.js
JScript Script File
91.6 KB

json2_min.js
JScript Script File
2.70 KB

FIGURE 1-1

Edge automatically creates several files and folders as you work. It's important not to move, delete, or rename these files, or your animation won't work as expected. You add your work to a website by copying these files to the web server.

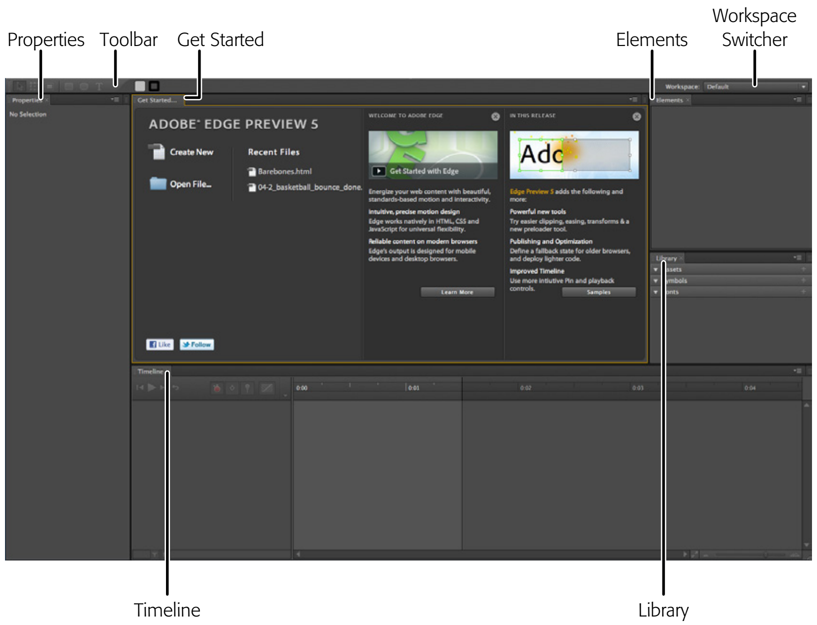
A Tour of the Edge Workspace

Fire up Edge for the first time and you see a workspace with several panels and a small toolbar, as shown in *Figure 1-2*. The name for each panel appears on a tab at the top. The Elements, Properties, and Timeline panels and the Tools toolbar all hold tools and widgets you use to create your animation masterpieces. The larger Get Started window initially offers some help and links to Adobe resources, but as soon as you open a project it turns into the *stage*, and the tab displays the name of your project.

- The **stage** is where you display and animate the graphics and text for your web page audience. When you save your project, Edge records the text and graphics and saves the description as a web page in HTML code. Open the page in a browser, and it plays back just as it appeared on the Edge stage. The stage has defined boundaries, and it's possible to hide or position elements so that they are offstage.
- **Elements** are objects that you add to the stage, and as a result they appear on your finished web page. Elements may be artwork, photographs, or text.
- Elements have **properties** that affect their position and appearance on the stage.

- The **Timeline** keeps track of elements and their properties over the course of time. When an element's properties change, that may change its position on the stage and its appearance.
- The **Library** keeps track of images that you import into your project. It provides easy access to the symbols that you create in Edge.
- **Tools** appear at the top of the main workspace. You use these to create, select, and modify elements on the stage. It's a small toolbox, but you may be surprised at how much it can do.

TIP You may think of these workspace boxes as panels or palettes, but Adobe lists them all under the *Window* menu, where you can show or hide each with a mouse click.



Energize your web content with beautiful, standards-based motion and interactivity.

Intuitive, precise motion design
Edge works natively in HTML, CSS and JavaScript for universal flexibility.

Reliable content on modern browsers
Edge's output is designed for mobile devices and desktop browsers.

Edge Preview 5 adds the following and more:

Powerful new tools
Try easier clipping, easing, transforms & a new preloader tool.

Publishing and Optimization
Define a fallback state for older browsers, and deploy lighter code.

Improved Timeline
Use more intuitive Pin and playback controls.

FIGURE 1-2
The Edge workspace consists of a few movable and resizable panels. Initially, the stage displays getting started info. After you create a new document, you use the stage as your canvas to display the graphic elements in your animation.

The Edge workspace takes its cues from other Adobe products. If you've used recent versions of Dreamweaver, Photoshop, or Flash, you'll feel right at home. If this is your first foray into Adobe territory, the techniques you learn here can be applied when you explore other applications.

Initially, all the panels are pieced together like a puzzle, but you aren't stuck with that arrangement. You can resize the panels within the workspace, or you can drag panels out so that they float independently. Want to make the Timeline bigger? To resize it while it's grouped snugly with the others, drag one of its edges. As it changes size, the surrounding panels change to accommodate the new arrangement. Want to move the Properties panel to a second monitor? Just drag its tab anywhere you want; the panel follows. If you have trouble freeing a window, click the small menu button in its upper-right corner (*Figure 1-3*) and choose Undock Panel. It will pop out from the main Edge workspace.

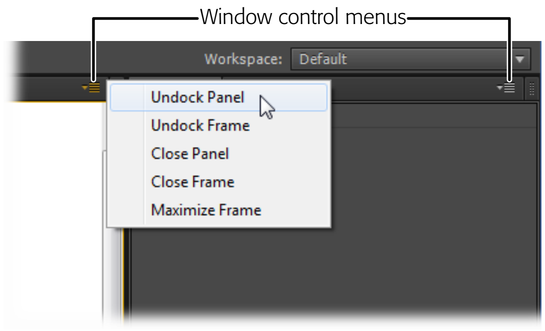


FIGURE 1-3

Use the menu in the upper-left corner of the individual panels to open, close, dock, and undock the panels.

Saving a Custom Workspace

Two scenarios may arise when you start dragging Edge's panels all over the place: Either you love the new layout or you hate it. Suppose you find the perfect layout for your work style and equipment. Perhaps you have a dual-monitor system and you like to have the stage fill one screen while the Timeline, Properties, and Elements panels camp out on the other. You can save the workspace layout using the Workspace menu in the upper-right corner. Initially, the menu is set to Default, as shown in *Figure 1-4*. Choose New Workspace, and a dialog box appears, where you can provide a custom name, such as "Dual Screen," for your custom workspace. Click OK, and now your newly named workspace joins the workspace menu. Just choose it whenever you want to use your handy Dual Screen workspace.

On the other hand, perhaps through dragging, tugging, and hiding panels you've arrived at a completely unworkable situation. You just want everything back the way it was when you started. Choose Default or any of the other workspace options, and all those panels jump back in place. Use the Reset option to return the currently selected workspace to its last saved arrangement.

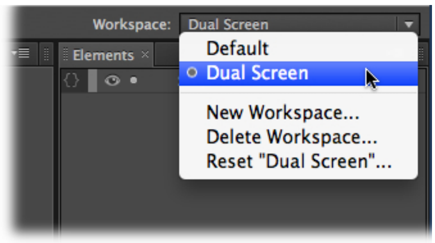


FIGURE 1-4

Use the Workspace menu to choose and create different layouts for the windows and panels you use in Edge. Use the New option to save your current workspace with a new name.

NOTE To help you keep your bearings, this book uses the Default workspace most of the time.

Building Your First Edge Animation

It's a long-standing coder's tradition to program a "hello world" test when first tackling a new language. In this case, Edge is going to write the code that displays your web page and animation, but why break with tradition? To dip your toe in the animation waters, you'll develop a hello-world page Edge-style. The blue marble of the earth will rise onto the stage, and the words "Hello World" will fade in and then fade out. You can use your own earth picture, or you can use *01-1_Hello_World* from *www.missingmanuals.com/cds/edgepv5mm*. The folder contains one image, *planet_earth.png*, which is used for this exercise.

If you want to see the final working example before you build it yourself, grab *01-2_Hello_World_done* from the Missing CD. Download and unzip it to find a folder that holds several files. You can view the completed project by opening *01-2_Hello_World_done.html* in any browser that's HTML5 capable. If you're not sure whether your browser can handle HTML5, see the box on page 23.

NOTE You can find all the examples for this book at *www.missingmanuals.com/cds/edgepv5mm*. Edge projects produce several different files and folders, such as HTML, JavaScript, and graphics, so the files for each exercise are in a folder. Individual examples are numbered. In the case of *01-1_Hello_World.zip*, the *01* at the beginning stands for Chapter 1 and *-1* indicates that it's the first exercise in the chapter. Completed examples for comparison are often included and have the word *done* in the filename, as in *01-2_Hello_World_done.*

1. Start Edge and go to File→New to create a new document.

   When you create a new document, the Get Started window becomes the stage. You see "stage" as the only element listed in the Elements and Properties windows. As you see in the Properties panel, the stage has dimension, color, and other properties. You'll learn more about each of these properties later.

2. Create a folder for your project and then choose File→Save As to save your file with a name like *Hello_World* or *First_Try.*

   You can create a folder outside of Edge using Windows Explorer or Finder, or you can create a new folder as part of a File→Save As command. It's a good practice to save your Edge project immediately with a helpful name. That way you won't end up with a bunch of "untitled" projects that you don't remember. Also, it makes it easy to save your work early and often with a quick Ctrl+S or ⌘-S. As explained on page 12, it's best to save each Edge project in its own folder because Edge creates several files and an *edge_includes* folder when you first save a project.

TIP A quick look at the Edge window tells whether your most recent work has been saved. If your work is unsaved, Edge shows an asterisk next to the filename at the top of the window.

3. In the Properties window, click the white Background Color swatch.

   A panel appears where you can choose a color (*Figure 1-5*). If you prefer a strictly visual approach, click the *spectrum* bar at the left for a basic hue and then click inside the square to fine-tune your selection. In some cases, you may have a specific color specification in RGB (red-green-blue) format or as a hexadecimal number. For more details on color management, see the box on page 28.

4. When the color picker appears, choose a dark blue color to represent deep space.

   If in doubt, try R=30 G=45 B=90 A=100 for this project. Edge uses Adobe's standard method for choosing numbers. When you see a highlighted number, that means you can either click and then type in a number, or you can click and drag to "scrub" in a number. Drag right to increase the number, left to decrease.

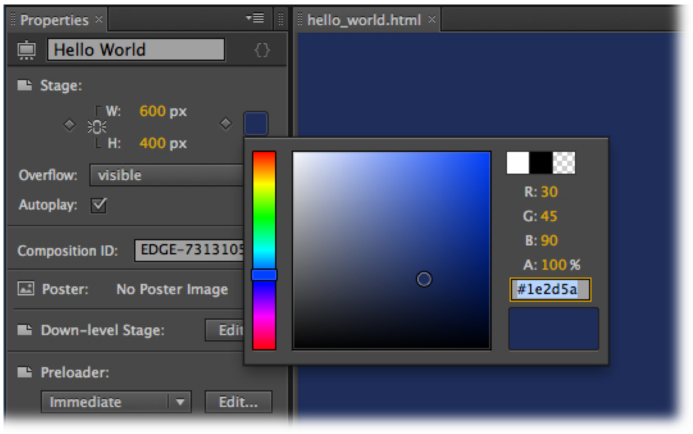


FIGURE 1-5

Edge uses the RGB (red-green-blue) color space used by most computer monitors and TV sets. The A stands for Alpha channel and controls opacity/transparency. The color picker lets you specify colors by pointing and clicking or typing in numbers.

5. Choose File→Import. Using the Import window that appears, find and select *planet_earth.png*. Click Open to import the image into your project.

   The *planet_earth.png* image was in the *01-1_Hello_World* from the Missing CD (*www.missingmanuals.com/cds/edgepv5mm*). After you import a file to your Edge project, it is listed in the Elements window and is displayed on the stage. It's automatically selected, so you see the properties for the newly imported element in the Properties window. The "planet_earth" has Location, Size, and Opacity properties. Below those, you see the Transformation properties that let you rotate, skew, and scale elements. Below that, the source file is listed—a handy point to keep in mind when you're trying to remember, "What the heck was the name of that file anyway?"

NOTE There's another bit of behind-the-scenes Edge magic going on here, too. When you import an image, Edge automatically creates an *images* folder in your project. It makes a copy of the image you select and puts the copy in the images folder. You'll also find your imported image listed under Assets in the Library panel.

6. In the Properties panel, click the ID box at the very top and change planet_earth to *World*.

   As Edge imports graphics, it names them using the file name. In some cases, that may be fine, but often you'll want to rename the element inside of Edge. Keep in mind this doesn't change the filename of your graphic. The ID World is used when you're working in Edge.

   IDs serve an important function in HTML code, as you'll learn later in this book. Notice that in the Elements panel your World appears with its new name. However, World is probably not yet listed in the Timeline. That's about to change.

TIP The names of animated elements automatically appear in the Timeline. You can choose whether or not non-animated elements are listed using the "Only show animated elements" button below the Timeline (see *Figure 1-6*).

7. In the Timeline, make sure the playhead is at 0:00.

   If you haven't made any Timeline changes since you created this project, the playhead is at 0:00, marking the first moment or frame of the animation, as shown in *Figure 1-6*. If you need to move the playhead, drag the gold-colored, bottom part of the playhead. The top part is called the *pin*. It should follow automatically. You'll learn more about the two-part playhead in the following steps.

8. Drag the World past the bottom of the stage.

   Items off stage appear a little darker. As you'll learn on page 26, you can control whether offstage items are displayed on the web page.

9. In the Timeline, make sure that the Auto-Keyframe Properties button is pressed.

   When the Auto-Keyframe Properties button (*Figure 1-6*) is pressed, keyframes are automatically created in the Timeline as you make changes to element properties. Keyframe markers look like diamonds. You'll learn all about keyframes and other Timeline features in Chapter 4.

10. Drag playhead to 0:01 on the Timeline.

    In the Timeline, 0:01 marks 1 second into the animation. A red line extends downward from the playhead, providing a marker for all the element and property layers.

11. With the World still selected, in the Properties panel, click the diamond next to Location.

    Two diamond-shaped keyframes appear in the Timeline marking the translate(x) and translate(y) properties. The X and Y properties set the position of elements on the stage. (Location properties are covered in detail on page 29.) By clicking the diamond next to Location in the Properties panel, you manually recorded the World's location on the stage. As a result, the World stays in the same X/Y position for the first second of the animation.

NOTE Edge automatically assigns a reference color to each element in your project. The color appears in the Timeline next to the name, and it's also used to display transitions—changes in property values. You see the same color next to the names in the Elements window. When you're dealing with dozens or hundreds of elements, the color coding comes in handy.

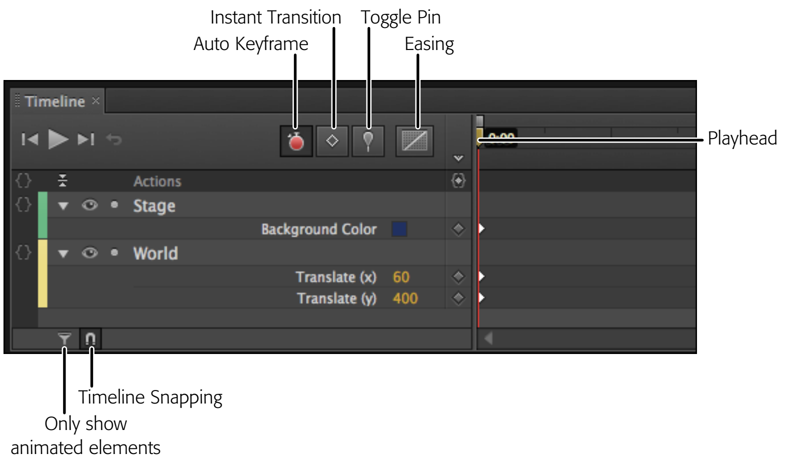


FIGURE 1-6

The Timeline keeps a running list of elements that appear on the stage. Not only that, it keeps track of their properties as they change. Here the Timeline lists the color of the stage and the position of the World graphic. Click the triangle next to the element names to expand and collapse the Properties lists.

12. Click the Toggle Pin button, then drag the playhead to 0:03.

    To animate an element, you change its properties over a specific period of time. The playhead and the pin let you mark two points in time, as shown in *Figure 1-7*.

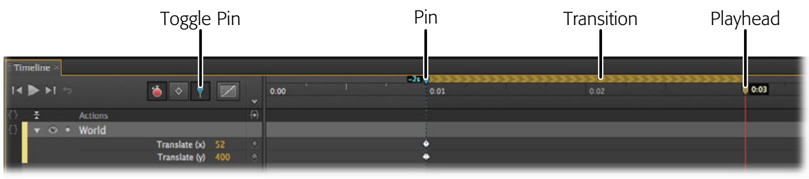


FIGURE 1-7

The pin "pins" the current properties at that point of time, while the playhead marks another point in time when the properties will be different.

13. Drag the World graphic so that Earth is visible on the stage.

    You can center the image on the stage, or you can choose some other eye-pleasing layout.

14. Press the Home key and then press the space bar.

    When you press Home, the playhead returns to 0:00. Pressing the space bar plays your animation so you can preview the action on the stage.

15. Move the playhead back to 0:00, and then in the toolbar, click the letter T.

    The text tool is selected, and the cursor changes to a cross.

16. Click on the stage and type *Hello World*.

    The words "Hello World" appear on the stage, but they're probably not positioned or formatted as you want.

17. In the Properties window, set the ID for the text box to HelloWorld.

    Naming your text makes it easier to identify in the Timeline and the Elements panel. Edge doesn't permit space in names, so you need to use HelloWorld or Hello_World.

18. Using the Properties panel, format the text.

    Change the text color to white or a very light blue. Choose Arial Black or another bold font. Adjust the size so it nearly fills the screen (72 px works well with Arial Black). Edge notes each change to the text in the Timeline, adding property layers and creating keyframes.

19. Position the text.

    If you're not sure about the placement, try it centered horizontally and about a third of the way down the stage.

20. With the playhead still at 0:00, set the opacity to 0.

    This means the text will not be visible at the beginning of the animation. Only the selection box shows and that will disappear as soon as you click something else. Don't worry, though—you can select any element, whether it's visible or not, by clicking its name in the Elements panel.

21. Turn Toggle Pin off.

    When Toggle Pin is off, the button doesn't appear pushed in and the pin moves with the playhead. The shortcut key for Toggle Pin is P. You'll use it a lot, so you might as well burn it in your memory now.

22. Drag the playhead to 0:02. With the text selected, click the diamond next to Opacity in the Properties panel.

    As you drag the playhead, you see the *World* move on the stage. Filmmakers and animators refer to dragging the playhead as *scrubbing*, a quick and easy way to review a segment of your animation. Clicking the Opacity diamond creates a keyframe at the 2-second mark where the text is still invisible.

23. Press P to toggle the pin, then drag the playhead to the 0:03 mark.

    Now you're ready to create another transition.

24. With the HelloWorld text box selected, set its opacity to 100.

    Edge creates a transition so that the text gradually changes from 0 to 100 percent opacity between 0:02 and 0:03 in your animation.

25. Drag the pin to the 0:03 mark, then drag the playhead to 0:04. Set the opacity back to 0.

    The text disappears again.

26. Press Ctrl+S (⌘-S) to save your work.

    As explained earlier, Edge saves your animation as a collection of HTML and JavaScript files. The main HTML file uses the name you provided in step 2, when you first saved your project. So, for example, you may see *Hello_World .html* in the project folder. When you imported the *planet_earth.png* image, Edge created an images folder and placed a copy of the graphic in the folder.

Your simple animation is complete. You can preview it in Edge by pressing Home and then the space bar. The earth rises into view, and your message fades in and then fades out (*Figure 1-8*). The entire animation takes 4 seconds.

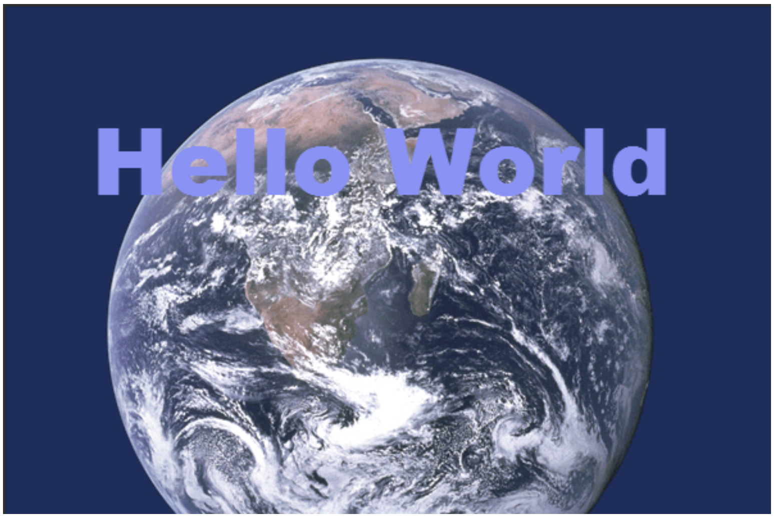


FIGURE 1-8

You can watch your entire animation inside of Edge by pressing Home and then the space bar. You can preview the animation in your web browser by choosing File→Preview In Browser.

Viewing Your Animation in a Browser

Your audience won't be viewing your animation in Edge; they'll be watching it in the familiar comfort of their favorite web browser. That means you need to review your work in a browser—preferably more than one browser. For a quick look, choose File→Preview In Browser. Edge starts your browser, if it isn't already running, and opens the HTML file that was created when you saved your project. That single HTML file describes the web page for your audience. All they have to do is load the page in a browser. The HTML code is actually the hub for all the other files the animation needs. It references the *planet_earth.png* image, which is stored as a separate file in the images folder. It also references the multiple JavaScript files needed to make everything run.

In animation, timing is everything. You may not be entirely pleased with the pace or other aspects of your Hello World experiment. In the coming chapters, you'll learn all about fine-tuning elements on the page so they look just right.

Here's a quick list of some important points to remember from this chapter:

- Edge creates multiple files and folders, so it's best to keep each project in a folder all its own.
- You can create and save a custom workspace that suits your work habits and your equipment.
- When you import a photo or graphic file, Edge creates a copy and stores that in the Images folder.
- Select an element on the stage or in the Elements panel and you see its properties listed in the Properties panel.
- To change the location or appearance of an element, select it and then change its properties. For example, change the location properties to move an element. Change the color or opacity properties to change its appearance.
- In the Timeline, keyframes record an element's properties at a given point in time.
- Animation occurs when properties change over time. These changes are marked by keyframes in the timeline.
- Transitions can be smooth (gradually changing over time) or abrupt.
- Handy keyboard shortcuts to remember: Home moves the playhead to the beginning of the timeline. Space plays the animation in Edge. Ctrl+Enter (⌘-Return) plays the animation in your web browser. The P shortcut toggles the playhead's pin, making it easy to mark two points in the timeline.

UP TO SPEED

HTML5 Browsers on the Leading Edge

The industry transition to full HTML5 compatibility isn't an overnight event. The features that make up the complete HTML5 specification are being implemented gradually over time in different browsers. If you want to test your browser's compatibility, head over to *http://html5test.com*. Free to use, this website provides a list of the HTML5 features your browser supports. Click the "other browsers" tab to see how your browser ranks with the competition.

If you're developing pages with Edge and expect a wide and diverse audience, you may want to test your work using several different browsers. The last couple of versions of Chrome, Firefox, Internet Explorer, and Safari would be a good start. You'll be pleased to know that most browsers in mobile devices are pretty HTML5 savvy.

There are some great resources for developers to learn more about browser capabilities. For example, *http://caniuse.com* displays a list of HTML5, CSS, and JavaScript elements. Click one of the elements listed, and you see a chart that explainswhich browsers and versions support the element. Another site, *http://html5please.us*, provides similar services.

CHAPTER

2

Creating and Animating Art

You can think of Edge as an animation management tool. Using Edge, you determine what elements show on the stage, their position, and their appearance. You can create text and simple visual elements within Edge, but it's likely that you'll create more complicated artwork in some other program like Illustrator, Photoshop, or Fireworks.

This chapter examines what types of graphics you can and can't create within Edge. It starts off by defining the stage and the ways you can modify it. You'll learn about all the properties of the rectangle and rounded rectangle. With creativity, you can create some distinctly non-rectangular shapes. Along the way, you'll learn how to quickly align and arrange objects on the stage. You'll test-drive the transform and clipping tools. The chapter then goes on to explain how to import artwork from your other favorite applications, such as Illustrator or Fireworks, and you'll get some tips about the best free graphics programs you can find on the Web.

Setting the Stage

As the Bard said a few hundred years ago, "All the world's a stage." That's certainly true for your Edge animations. As explained in Chapter 1, when you place an element on the stage, it's visible to your audience. There are a couple of ways to hide or remove elements from the stage. If you have the stage Overflow properties set to hidden, then you can exit stage right, left, top, or bottom by moving the element off stage. At least, it's not visible. The Hello World exercise also showed how to perform disappearing acts by using the Opacity property.

The stage that you work with in Edge represents a portion of a web page when it's viewed in a browser. The stage has a limited number of properties. The most obvious are its dimensions and background color, but you'll want to understand them all. Here's the rundown starting from the top of the Properties panel:

- The **Document Title**, as you might guess, is the name of your animation. When you save a project, Edge creates a web page, also known as an HTML document. Most browsers show the title of the web page in a tab or the window's title bar.
- Stage **dimensions** are shown as W (width) and H (height) properties. No big surprises here. You can type in or scrub in the width and height of the stage. The stage doesn't have to appear in the upper-left corner of a web page. On page 148, you'll learn how to reposition the stage. For example, if your Edge animation is a banner ad, you might create a tall, narrow stage and then position it on the left side of the page.
- The **background color** is set using a color picker. In the Properties panel, click the color swatch and a color picker appears, as shown in *Figure 2-1*. Click the bar (also called the *spectrum*) on the left to choose a hue, and then click in the larger square to fine-tune the shade. The circle is positioned over the selected color, and the swatch in the lower-right corner displays it. The three swatches at top right make it easy to quickly choose a white, black, or transparent background. If you work with a team, you may be given a color spec in RGB or hexadecimal formats. On the other hand, if you're calling the shots, you may want to specify a color for other designers. For the details on specifying a color by numbers, see the box on page 28.
- The **Overflow** menu controls the way elements appear when they're offstage. Often, you'll want to set this menu to *hidden*, which makes elements outside the stage's rectangle invisible. The hidden option works well when you want to have elements enter and exit the stage. If you set the menu to *visible*, elements that move beyond the boundary of the stage remain visible as long as there's room on the web page. The *scroll* option places scroll bars at the right and bottom of the stage, making it possible to view elements that move outside the specified dimensions of the stage. The *auto* option automatically adds scroll bars if content exists beyond the confines of the stage.
- Use the **Autoplay** checkbox to tell your animation to automatically run when its web page is loaded in a browser. If the box is turned off, you must use a JavaScript trigger to run the animation.
- The **Composition ID** is used to identify this particular stage and its accompanying timeline. This becomes important when you have more than one Edge animation on a single web page. You'll learn more about this in the JavaScript chapters.
- The **Poster** and **Down-level Stage** properties create alternative elements for web browsers that aren't HTML5 savvy.

- The **Preloader** is responsible for loading all the resources needed to display your composition on a web page. Those resources include JavaScript libraries and graphics. You'll learn more about these controls on page 144.

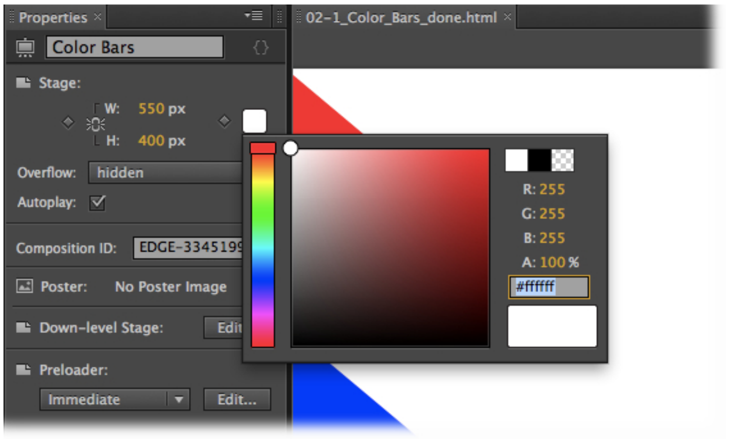


FIGURE 2-1

Edge presents the same color picker whether you are choosing the color for the stage, text, or a drawn object. The selected color shows in the lower-right corner. You can dial in your color visually or by typing in a color spec.

Creating Art in Edge

OK, admit it. When you first saw the Edge tool palette, you said, "What? A rounded rectangle but no circle?" In its current incarnation, Edge's drawing tools are limited, but in this section you'll at least learn how to create circles and ovals. Why is art so primitive in Edge compared with other Adobe products such as Flash or Illustrator? Part of the reason is that Edge is new. It's likely that Adobe implemented the graphics features that were easiest first and that Edge will become more full featured over time. Keep in mind that every time you create a graphic object in Edge, it's busy behind the scenes writing code in JavaScript to describe that object. If you're a conspiracy theorist, you might think Adobe wants you to spring for Illustrator or one of the other artist's tools it sells in its Creative Suite. In fact, if you want to create vector graphics with complex paths, you need to use a tool like the pay-to-play Illustrator (*www.adobe.com*) or the open source Inkscape (*www.inkscape.org*).

The next section describes in detail the properties of the rectangle. However, many of these properties are used by other objects, such as blocks of text and artwork that you import into Edge. So when you're learning all about rotating, skewing, and scaling rectangles, keep in mind that you can rotate, skew, and scale text and photos, too.

TIP If you're on a budget, you may want to use one of the free, open-source applications that create art. Inkscape (*www.inkscape.org*) is a vector-based drawing program (similar to Illustrator), and GIMP (*www.gimp.org*) is a raster-based photo editing application (similar to Photoshop).

GEEK SPEAK

Understanding Color Specs and the Color Picker

If you're new to digital art, the various ways to specify color may seem confusing. As an artist, you know that color doesn't exist in a vacuum. The colors you get when you mix pigments aren't the same as the colors you get when you mix different colored lights, which is how a computer monitor works. Artists working in oil paint or pastel use the red-yellow-blue color model, while commercial printers use the cyan-magenta-yellow-black model. In the world of computer graphics, the color model you use is red-green-blue, or RGB.

The Edge color picker (Figure 2-1) has three different ways to reference a color using the RGB color model. You can pick a color visually using the spectrum bar and the square next to it. As you do so, different numbers appear on the right. The R, G, and B numbers represent the quantity of red, green, or blue added to the mix. The numbers range from 0 (0 percent) to 255 (100 percent). Why the odd numbers? Computers like powers of two and 256 (2^8) falls in that category. Computers also start their counting at zero. As a result, RGB colors use a scale from 0 to 255 to mix colors. The advantage to you, the human artist, is that you have more precision in identifying colors than if you used 0 percent to 100 percent.

So what about that A? The A stands for Alpha channel, which controls the overall opacity for the final mixed color. When A is zero, the color is completely transparent. You can use the Alpha channel whether you specify a color using the visual tools or one of the number systems.

In the coder's world, there's another common way to specify a color, and that's done with hexadecimal numbers. Hexadecimal numbers are base 16 instead of the base 10 numbers people usually use. (How's that for a flashback to math class?) Hexadecimal numbers translate more easily into binary numbers than our familiar decimal system. The hexadecimal number system uses 16 symbols (2^4) to represent numbers instead of the usual 0-9. When the common numeric symbols run out, hexadecimal uses letters. So the complete set of number values looks like this: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

To specify an RGB color with hexadecimal numbers, you use six places. The first two numbers represent the amount of red in the mix, the second two numbers represent green, and the final two numbers represent blue. So a color specification might look like this: 0152A0. Or this: 33CCFF. At first, it seems odd to see letters in numbers, but after a while you get the hang of it. So the hexadecimal FF0000 is a bright, pure red, while 0000FF is a bright blue.

In Edge, hexadecimal numbers are differentiated from base 10 numbers by placing a pound sign (#) in front. That's the way the color picker shows the hexadecimal number for your color selection, as you can see in Figure 2-1.

Rectangles: Building a Basic Box

Using the Rectangle tool (M), you can add blocks of color to the stage. These blocks are great if you want to differentiate portions of the web page. For example, perhaps you want to make a sidebar. Add a rectangle, and then you can place text or graphics over the rectangle, setting it off from the rest of the page. Chances are you know the basic drill for creating a rectangle. Click the Rectangle tool on the Tools palette, and then click and drag on the stage to mark its shape. To create a square, hold the Shift key while you drag. The new element appears on the stage, and it's automatically selected, so you see eight white squares around the border that represent handles (*Figure 2-2*). You can continue to change the size and shape of the rectangle after it's drawn by dragging the handles. Here are the basic properties that describe your rectangles:

- **ID.** As soon as you draw a rectangle on the stage, it's listed in the Elements panel. When the rectangle is selected, its properties appear in the Properties panel. As with all your Edge elements, you probably want to give your rectangle a meaningful ID, such as LeftSidebar or Header. Otherwise, you'll be searching through Rectangle1, Rectangle2, and Rectangle3 trying to find the one you want. To rename your rectangle, select it and change the ID at the top of the Properties panel. As an alternative, you can double-click the name in the Elements panel.
- **Element Display.** Some elements are always on stage while others may come and go. The Element Display menu gives you a way to easily hide an element until it is needed. Your three choices include: Always On, On, and Off.
- **Tag.** Your rectangle is automatically assigned an HTML <div> tag. Edge uses these tags to identify, position, and transform elements. Chapter 7 covers HTML code in detail.
- **Location.** Underneath the name in the Properties panel, you see the Location and Size properties. The upper-left corner of the stage is referenced as X=0, Y=0. Moving from left to right increases the X value. Moving from top to bottom increases the Y value. Your rectangle's position is referenced by the upper-left corner. It's position on the stage is shown under Location as X and Y properties.
- **Size.** Next to the Location properties are the Size properties: W (width) and H (height). These change automatically when you drag a rectangle's handles. You can also type in or scrub in a specific number. For rectangles, the size properties use pixels as the unit of measure. Text boxes also let you use ems, a typographer's form of measurement. An em is the width and height of the letter M, so it is considerably larger than a pixel. Use the link next to the W and H properties to lock and unlock the aspect ratio for your rectangle. When the Link Scale is unbroken, changing one dimension automatically changes the other so that the rectangle stays proportionate. When the link is broken, you can change W and H independently.
- **Opacity.** Use the slider under Location and Size when want to control the Opacity of the entire rectangle. When you want to adjust the opacity of the border or background independently, click their color swatches (explained under Color) and change the A (alpha property).
- **Overflow.** The overflow control for your rectangle works like the one for your stage, except it explicitly applies to the rectangle.
- **Color.** Rectangles have two basic parts: border color and background color. Border color marks the outer edge of the rectangle, while background color is the color inside the box. (Other programs sometimes call these properties *stroke* and *fill.*) You can assign separate colors to the border and background, or you can make them transparent by setting the Alpha channel to zero. There are two additional properties for the stroke. You can set the size in pixels (px) and you can choose among a solid stroke, a dashed stroke, or none—no stroke at all.

TIP As this was written, Edge didn't offer any way to create color gradients. That's where one color blends into another color. These effects are standard tools for graphic artists. If you want to use gradients, you'll need to create your art in Photoshop, Illustrator, or Fireworks and then import it into Edge. If you want to create a gradient that blends to transparency, letting the background show through, use the PNG32 graphic format.

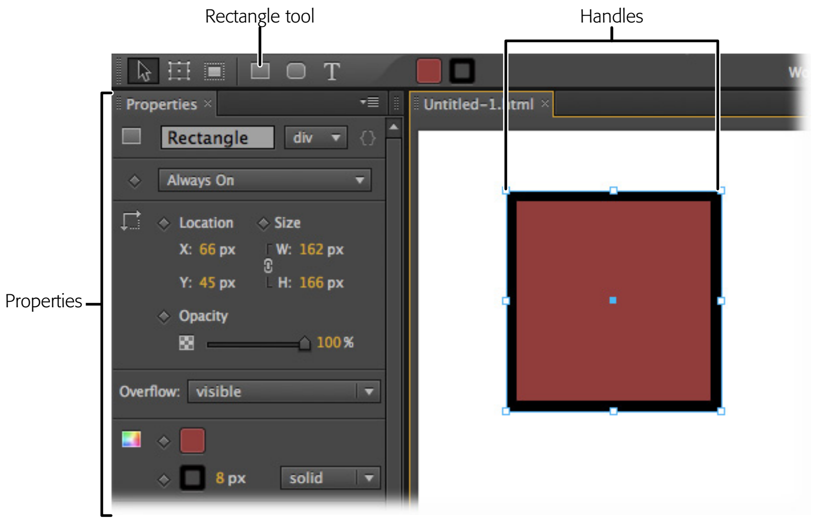


FIGURE 2-2

After you create a rectangle, you can change its shape and size by dragging the handles. When a rectangle is selected, its properties appear (where else?) in the Properties panel.

Selecting and Copying Elements

When it comes to selecting, cutting, and pasting objects, Edge works like most other computer programs. You use the same techniques for objects you create in Edge, like rectangles and text, or artwork that you import (as explained on page 42). You use the Arrow in the Tools palette to do the heavy lifting. Click once on an element to select it. Shift-click on another element to add it to the selection. Edge has a handy command (Ctrl+A or ⌘-A) to select everything on the stage. You can easily deselect objects by clicking on an empty spot around the edge of the stage.

Edge also provides the usual suspects when it comes to Cut (Ctrl+X or ⌘-X), Copy (Ctrl+C or ⌘-C), and Paste (Ctrl+V or ⌘-V). The Duplicate command (Ctrl-D or ⌘-D) combines copy and paste into one function. When you're in need of several carefully rounded and shaded rectangles, it's much easier to create one and then clone it with Duplicate. (If you prefer menus to shortcut keys, you'll find all these commands on the Edit menu.)

When you duplicate an element that's on the stage, you also duplicate any transitions that may have been created for that element. If you don't want the transitions, use Copy (Ctrl+C or ⌘-C) and Paste (Ctrl+V or ⌘-V). You'll find details for editing transitions on page 86.

TIP Even though you can quickly create new elements using the copy-and-paste technique, you may still want to give each new element a unique, useful name. All you'll get from Edge are less-than-helpful names like RectangleCopy and RectangleCopy2.

Transforming Your Rectangle

The Transform properties, along with the Transform Tool (Q), are there to help you fold, spindle, and mutilate your rectangles. Well, actually the transforms are called Rotate, Skew, and Scale. The other property in the Transform group is called Transform Origin. You can think of the origin as an anchor point for your rectangle. It appears as a small blue square when the rectangle is selected. Initially, the origin is at the midpoint. So if you rotate the rectangle, it spins around the midpoint. If you move the transform origin to the bottom-right corner by setting X=100% and Y=100%, your rectangle will rotate around that bottom-right corner point. Using the X and Y Transform Origin properties, you can move the origin to any point in your rectangle. As you make adjustments, the small blue square moves over the surface of the rectangle. If you don't mind eyeballing it, choose the Transform Tool (Q) and then you can click and drag the transform origin to a new point as shown in *Figure 2-3*.

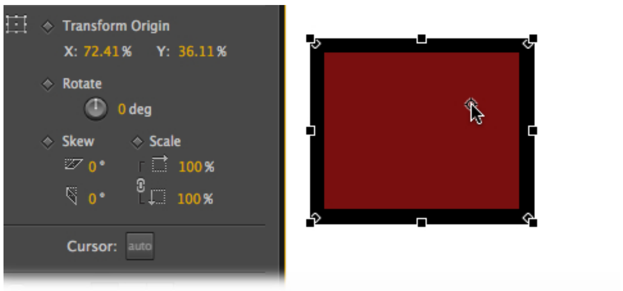


FIGURE 2-3

The transform origin point is the center of the universe when it comes to rotating, skewing, or scaling elements on the stage. You can reposition the transform origin using the X and Y properties shown here.

- **Rotate.** The Rotate properties are straightforward. When you create an element, like a rectangle, the Rotate property starts out at 0 degrees. You can manually rotate an object with the Transform tool (as shown in *Figure 2-4*) or you can use the Properties panel. If you want to spin your rectangle, change the Rotate property. If you want the spinning motion to take place over time, you need to create two keyframes in the Timeline at different points in time with different Rotate properties. (For more details on the Timeline and keyframes, see page 73.) If your first rotation keyframe is set to 0 and your second is set to 360, the rectangle spins in a complete circle. Set that second keyframe to 720, and it spins twice.

- **Skew.** The Skew properties turn your rectangle into a parallelogram by sliding two opposing sides in different directions (*Figure 2-4*). The top Skew property slides the top and bottom sides, and the bottom Skew property slides the left and right sides. You can apply both types of skew to a single object. The best way to get the hang of the Skew properties is to jump in and start playing with it.
- **Scale.** When you scale a rectangle, you change its size—making it smaller or larger. The Scale properties represent a percentage of the original object in the horizontal and vertical direction. As with the Size properties (page 29), the link to the left of the numbers lets you choose whether or not the scaling is proportionate.

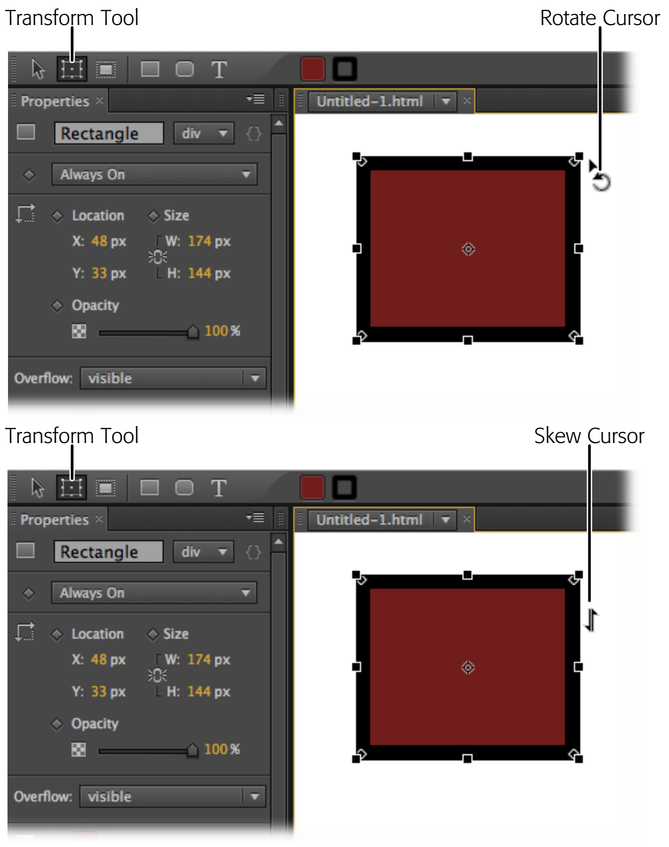


FIGURE 2-4

You can make your transformations using the Properties panel or you can use the Transform tool for a hands-on approach. After selecting the Transform tool, move the cursor over the element you want to change, the cursor changes to indicate the transformation that's about to be performed.

Top: With the cursor near the corner you see Rotate cursor.

Bottom: With the cursor mid-slide, you see the Skew cursor.

Aligning, Distributing, and Arranging Elements

The maxim that "everything has its place" is certainly true when it comes to animation. With more than one element on the stage, their relationship to each other is critical. Designers often have a specific grid in mind when they're creating printed pages or web pages. It's best when boxes of text or graphics are aligned with this invisible grid. When several elements are aligned, it usually looks best when there's an equal distance between them. You can spend a lot of time eyeballing the stage to try to get everything just right, but fortunately, you don't have to.

To experiment with Edge's Arrange, Align, and Distribute tools, you may want to create three or four simple objects from the Rectangle and Rounded Rectangle tools like the ones shown in *Figure 2-5*. As you drag elements around the stage, you'll notice magenta-colored lines sprouting from the edges and midpoints. These are Smart Guides, and they can help you to quickly align one or more objects while you're mid-move. In many cases, that may be all the help you need.

TIP If you find the Smart Guides distracting, you can toggle them on and off as needed with the Ctrl+semicolon or ⌘-semicolon.

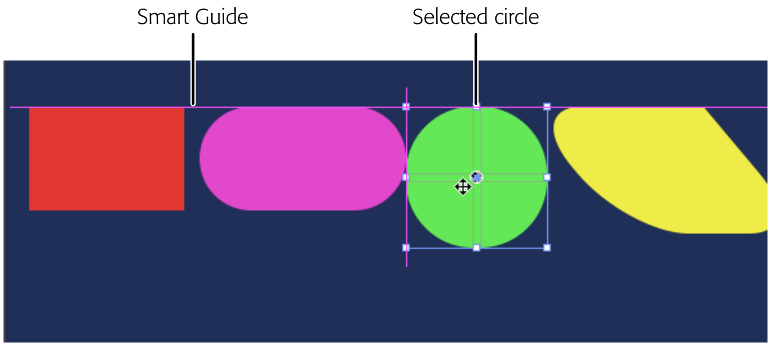


FIGURE 2-5

Smart Guides are smart enough to hide when you don't need them. However, they show up when there's a job to do. Here the Smart Guide appears to help align the top edge of these four shapes. You can use guides to align any edge or the midpoint of an element.

For more formal alignment needs, turn to the Modify→Align menu. For align to work, you need to select at least two elements. One of those elements may be the stage. You can select the elements on the stage or you can use the Elements panel. To use these commands, select all the elements that you want to align and then choose one of the options:

- Modify→Align→Left
- Modify→Align→Horizontal Center
- Modify→Align→Right

- Modify→Align→Top
- Modify→Align→Vertical Center
- Modify→Align→Bottom

You use the Modify→Distribute commands to put equal distance between three or more elements on the stage. You can choose which part of your elements the distribute command uses for the process and whether the action takes place along the horizontal or vertical axis. The specific commands are:

- Modify→Distribute→Left
- Modify→Distribute→Horizontal Center
- Modify→Distribute→Right
- Modify→Distribute→Top
- Modify→Distribute→Vertical Center
- Modify→Distribute→Bottom

Arranging Elements: Z-Order

In addition to horizontal and vertical position, there's another way you can arrange objects on your stage. As you create elements, you may notice that new elements appear to be in front of the older elements, and if you drag a new element to the same X/Y position on the stage, it hides an older one. If you're familiar with Photoshop, you might think of this positioning as "layers." In geek-speak, it's often referred to as the Z-layer or the Z-order, because this third dimension is known as the Z axis. You can examine the Z-order of the elements on the stage by simply looking at the Elements panel. Elements at the top of the list are closer to the front. If you want to change the order, just drag an element to a new position in the panel. Edge also gives you menu commands and shortcut keys to rearrange elements:

- Modify→Arrange→Bring to Front (Ctrl+Shift+] or Shift-⌘-])
- Modify→Arrange→Bring Forward (Ctrl+] or ⌘-])
- Modify→Arrange→Send Backward (Ctrl+[or ⌘-[)
- Modify→Arrange→Send to Back (Ctrl+Shift+[or Shift-⌘-[)

A Rectangular Animation

Roll up your sleeves. Enough theory, it's time for some animation. In this exercise, you create four rectangles. You give them names, apply color, and skew them. Then you position them on the stage and make them move, change shape, and then appear to dissolve. It's the sort of effect that might be part of a banner ad or the introduction to a more complex animation. You don't need any Missing CD files to tackle this exercise. However, if you want to see the finished project, check out *02-1_Color_Bars_done.zip*.

This exercise is divided into two parts. In the first set of steps, you create and position the color bars:

1. Open and save a new Edge project with the name *Color_Bars*.

   Don't forget to create a new folder for your project.

2. Set the stage color to white and the dimensions to W=550 px and H=400 px.

   Edge remembers the last stage settings you used. So if you followed previous exercises or experimented on your own, you may need to make these changes.

3. In the Timeline (*Figure 2-6*) make sure that the Auto-Keyframe Properties button is pressed.

   If you move your cursor over the buttons, tooltips show their names. For example, at the top of the Timeline you see: Auto-Keyframe Properties, Instant Transitions, Toggle Pin, and Easing.

4. Draw a rectangle and name it *Red*.

   The Name box appears at the top of the Properties panel when the rectangle is selected.

5. In Properties, click the background color and set it to pure red, and set the border to *none*.

   When you're done, the hex color number should be #ff0000.

6. Set the rectangle's size to W=550 and H=100.

   The quickest way to accomplish this is to type the dimensions in the Properties panel, but if you're a mouse master, you can drag the rectangle's handles. You may need to click the "link" button next to W and H to change the width and height independently.

7. Set the Skew (x) to 50 deg (degrees).

   The horizontal skew is the top setting. A positive number slides the top edge to the left and the bottom edge to the right.

8. Position your red, skewed rectangle in the top-left corner of the stage so that only its point tip is visible. The Location properties should be X=-550, Y=0.

   Ideally, just a red triangle shows in the top-left corner of the stage.

9. With the Red rectangle selected, press Ctrl+D (⌘-D). Change the name of RedCopy to Green. Then, change the color to match.

   The hex value for solid green is #00ff00.

TIP The Duplicate command Edit→Duplicate is identical to the two step process of choosing Copy (Ctrl+C or ⌘-C) and then Paste (Ctrl+V or ⌘-V). In either case, you have a new copy of the element with the word "Copy" tacked onto the end of the name.

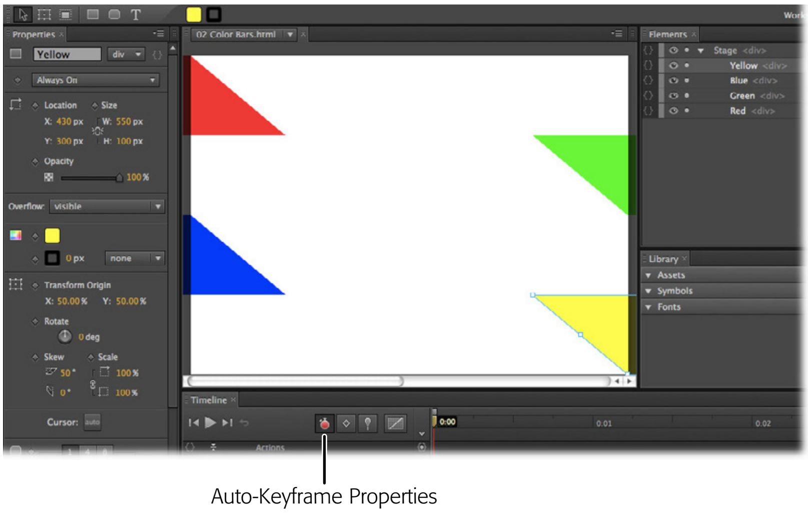
10. Line the top of the green rectangle with the bottom of the red rectangle (Y=100). Then, hold the Shift key down and slide Green to the right until only the tip shows (X=430).

    Holding the Shift key down while you move an element helps to lock it to the horizontal or vertical axis as you drag it. You can still drag it off axis, but it's a little "sticky."

11. Create two more skewed rectangles, naming and coloring them Blue and Yellow. Position the rectangles on alternating sides of the stage.

    The blue color is #0000ff, and yellow color is #ffff00. When you're done positioning the rectangles, the stage should look like *Figure 2-6*.

TIP If you later have difficulty animating these elements, draw the last two triangles instead of duplicating them. Some versions of Edge get confused when you duplicate an element more than once. Most likely, this issue will get resolved as Edge matures.



Yellow div
Always On
Location X: 430 px Y: 300 px
Size W: 550 px H: 100 px
Opacity 100%
Overflow: visible
0 px none
Transform Origin X: 50.00% Y: 50.00%
Rotate 0 deg
Skew 50° 0°
Scale 100% 100%
Cursor: auto

FIGURE 2-6

At the outset of the Color Bars animation, the four skewed rectangles start off on opposing sides of the stage. The animation makes them move across the stage and change size and shape.

Animating by Adding Property Keyframes

Now that you've successfully created and positioned the color bars, it's time to make them move. Chapter 1 showed how the position of elements on the stage is controlled by property keyframes in the Timeline. When the Auto-Keyframe Properties button is pressed, as shown in *Figure 2-7*, new property keyframes are created whenever you set or change a property. You can also create property keyframes manually by clicking the diamond-shaped buttons in the Properties panel. You want to lock in the Location, Size, Color, and Skew properties at the beginning of your animation by creating property keyframes. Then you'll move down the Timeline and create different property keyframes. The result will be animation magic.

1. Make sure the Timeline's playhead is at 0:00. Select the parallelogram named Red; then in the Properties panel, click the diamond-shaped buttons next to Location, Size, Opacity, and Background-Color.

   The diamond buttons add property keyframes and individual property layers in the Timeline as you can see in *Figure 2-8*. Property keyframes anchor a specific property value at a specific point in time. In the Timeline, you should see property keyframes and property layers for:

   - Translate (x)
   - Translate (y)
   - width
   - height
   - Opacity
   - Background Color

   If you don't see all those keyframes and property timelines under Red, you should create them manually by clicking the diamond button next to the missing property.

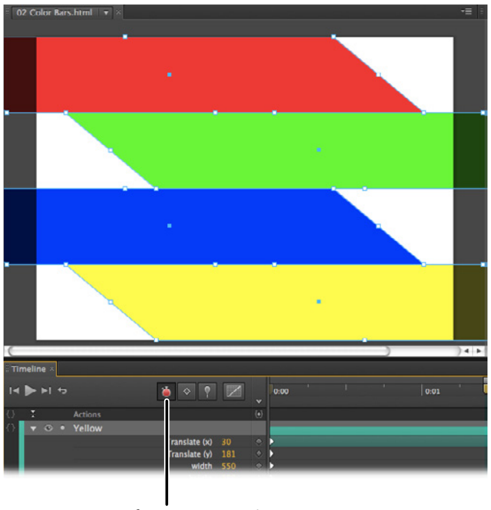


FIGURE 2-7

Keyframes anchor an element's property to a specific value at a specific point in time. Here Location, Size, Color, and Skew properties are set for the Yellow element when the Timeline is at 0:00.

TIP You can expand and collapse the lanes (property rows) for each element by clicking the triangle button next to the element name. A button at the top, left of the word Actions, collapses or expands all lanes.

2. Repeat step 1 for the Green, Blue, and Yellow color bars to create the property keyframes and property layers for each.

   To speed things up, you can select all three bars first and then click the keyframe buttons.

3. Make sure the Auto-Keyframe button is on (pressed in) and the other buttons are not.

   When Auto-Keyframe is on, Edge automatically creates property keyframes as you change elements on the stage (*Figure 2-8*). It's a two-step process. Move the playhead to a point in time and then change your element's properties. You can make changes in the Properties panel or you can make changes on the stage with the Selection and Transform tools.

4. In the Timeline, drag the playhead to the 0:02 position.

   For this step, the pin should be toggled off (not pressed in).

5. Drag each of the rectangles across the stage until the tail end of the skewed rectangle is visible.

   At this point, most of the stage is covered by the color bars, with white triangles of the stage showing through at the edges. Remember to press the Shift key as you drag if you want to steady the bars' vertical position.

6. With all the rectangles selected, in Properties click the Add Key Keyframe for Opacity button. Drag the playhead back and forth to preview the animation.

   The opacity for each color bar is set to 100 percent at the 2 second point. Scrubbing the playhead gives you a quick look at the action.

7. Drag the playhead to the 0:03 marker.

   This position represents the point 3 seconds into your animation.

8. Select each rectangle and then change the height (H property) to 300 px and the opacity to 50 percent.

   This has the effect of making the rectangles grow, slicing vertically into one another, and at the same time start to blur. Keep in mind, you may need to delink the W and H properties to change them independently.

9. Drag the playhead to the 0:04 marker. Then change each rectangle's height to 500 px and the opacity to 0 percent.

   The effect is that the rectangles keep growing and blur out of view.

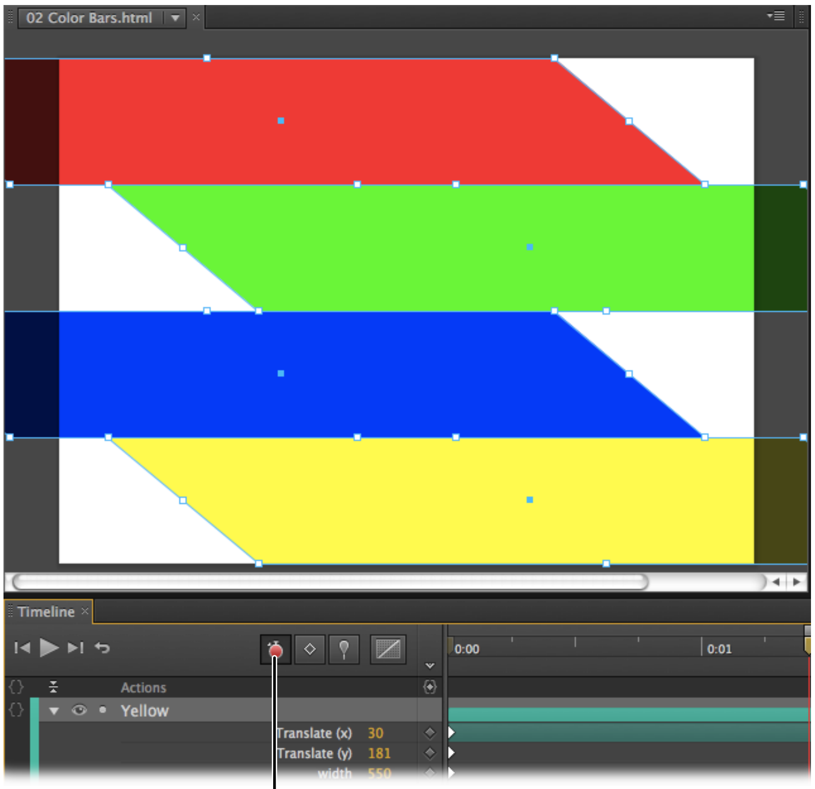


FIGURE 2-8
The Timeline keeps track of property changes over time. When the Auto-Keyframe Properties button is depressed, the Timeline automatically creates property key-frames when you set or change a property. If it is not set, you need to manually create property keyframes by clicking the diamond-shaped buttons next to properties in the Properties panel.

Reviewing and Troubleshooting Your Animation

When you're finished with the exercise, it should look like *02-1_Color_Bars_done*. To watch your animation, press Home and then the space bar or use the controls on the Timeline.

If the animation doesn't play as expected, the most likely culprit is a missing or misplaced property keyframe. Move the playhead to 0:00 on the Timeline and check to make sure each color bar has the correct property values for:

- Translate (x)
- Translate (y)
- width

- height
- Opacity
- Background Color

It's a common mistake to assume that a property value is set at the start of an animation when it isn't. If there are still gremlins in the machine, compare the values at the 0:02, 0:03, and 0:04 positions with those in the Missing CD example file.

If you're in the mood to experiment, try staggering the bars' movement over time or changing the size, color, or skew properties in different ways. Add a background image or text to make the animation more interesting and the transparency revealing.

Rounded Rectangles: More than Meets the Eye

OK, Edge pulls a fast one when it comes to the Rectangle and Rounded Rectangle tools. The dirty little secret is that you can create a rectangle with either tool and they're identical. You can check this by creating a rectangle with each tool and examining their properties. You can turn a rectangle into a rounded rectangle simply by adjusting the *radii* properties at the bottom of the Properties panel, as shown in *Figure 2-9*. Likewise, you can square off a rounded rectangle using the same tools. So here's a look at how they work.

In a new Edge project, create a rectangle and leave it selected. Choose the Transitions Tools (Q), then with the mouse, hover over the keyframe diamond in the rounded rectangle properties, and the tooltip explains that it will "Add Keyframe for Border Radii." The three buttons at the top of the panel are labeled 1, 4, and 8. Below, you see a square made up of buttons where you can individually select the four corners of a rectangle. There's a number next to the corner buttons that's initially set to 0. A corner radius of zero means your rectangle has nice, sharp-edged corners. Click on the number and drag to the right to round off the corners. The number box accepts only positive numbers, so you can't drag left. Notice that as you drag, the black diamonds at the corners of your rectangle move to the center. These diamonds are control points for the corner radii. You can manually drag the diamonds on any rectangle to create and adjust rounded corners.

Reset your rectangle so that it's square, and then click the upper-right corner in the Properties panel. Change the radius setting and this time, you notice that the upper-right corner remains square while the others take on the rounded style. When a corner button is pressed in, that deselects the corner from the rounded settings.

Reset the rectangle once more and click the upper-right corner so it pops back out. Then click the 4 button at the top of the corner properties. Four new number boxes appear next to each corner. Now you can set each corner independently with different radius values. This gives you the ability to create irregular shapes even though, technically, they still have four corners. Combine this with the skewing and scaling properties, and you can create some really interesting amoeba effects. Click the 8 button, and each corner has two control numbers. This gives you the ability to move the control point off center, making a corner that is flatter on one side compared to the other. Notice that when you adjust the settings, the black diamond control point moves, too. You can always adjust your corners using either the number boxes or the control points in the rectangle.

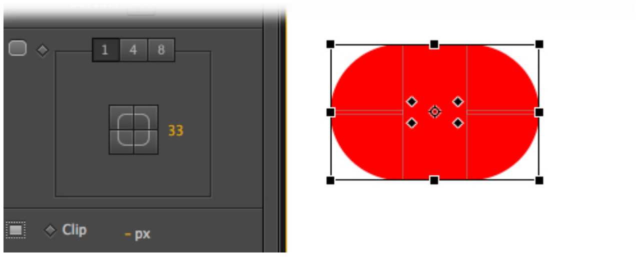


FIGURE 2-9

You turn a plain old rectangle into a rounded rectangle using these settings found at the bottom of the Properties panel. You can round all the corners the same amount, or you can use different settings for each.

A Circle Is a Very Rounded Rectangle

It's not unusual to want to create a circle or oval, so it might seem odd that Edge doesn't provide a tool to do that. However, you can use your Rounded Rectangle tool to create circles and ovals. To create a circle, start with a square.

1. Click the Rectangle tool and, while holding the Shift key, drag out a box.

   Holding the Shift key down constrains the rectangle so that all sides are equal.

2. In the rounded corner properties, click the 1 button.

   With this setting, all the corners share the same corner radius value.

3. Click and drag the border radius number box until the black diamonds all meet in the center, as shown in *Figure 2-10*.

   It's possible to drag the number so that the corner radii pass one another at the center, but that's not necessary to create a circle.

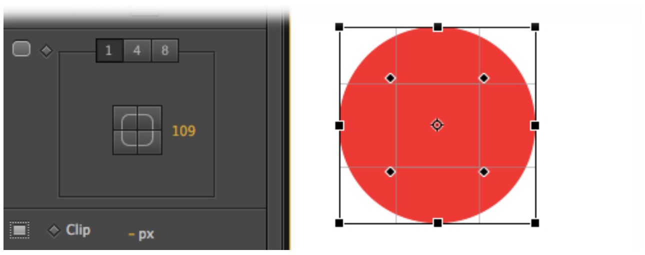


FIGURE 2-10

You can create circles with the Rectangle or Rounded Rectangle tools. Start off by making a square, and then add a radius to all four corners. Using similar techniques, you can create ovals and ellipses.

You can change and adjust your circle properties just as you would any other object that you create in Edge. By skewing your object, you can create ellipses. By scaling it, you can create ovals. And, of course, you can create anything in between a square and an ellipse with the right settings.

Importing Art

It's easy enough to create basic shapes and text in Edge, but when it comes to complex artwork, you'll probably turn to your favorite art creation tools. For elaborate drawings and line art, that may be Adobe Illustrator. For photographs, you may use Photoshop, Lightroom, or iPhoto. No matter how you create JPEGs, SVGs, PNGs, or GIFs, you can import them into Edge and then animate them by changing their position on the stage and their appearance.

There's one difference in the way Edge handles artwork that you create using the built-in tools, like the Rectangle tool, and art that's imported. Art created with the built-in tools is created on the fly with JavaScript code, while imported art is stored in its native format (JPG, SVG, PNG, or GIF) in the images folder. Those images are referenced by the HTML and JavaScript code. If the files themselves are renamed, moved, or missing, your audience is going to miss part of the show.

Regardless of the file format, the process for importing artwork is the same. Go to File→Import and then find the file you want to bring into your project. The Missing CD folder *02-2_Sliding_Show* has three photos in JPEG format. You can practice by creating a new project complete with a new folder named *02-2_Sliding_Show* and import each of the photos. After you choose File→Import, a standard file/folder window opens for your PC or Mac. If you want to import all three files at once, just Shift-click to select them. As usual, Edge imports the files and, as a handy timesaver, names them based on the filenames. In this case, you'll find squirrel, farmhouse, and bike in your Elements panel. Each image is also automatically placed at the 0,0 position on the stage. You'll only see one of the images though, because they're covering one another. Remember that Z-order stuff on page 34?

Showing and Hiding Elements

Sometimes you may have so many elements on the stage that you can't find or identify the one you want. The Elements panel can help you isolate and identify elements on the stage by showing and hiding them. To the left of each filename is an eyeball toggle button (*Figure 2-11*). When the eye is displayed, the object is visible on the stage. Click the eye and it turns into a circle; the element is hidden from view in Edge. You can identify each of the imported photos by temporarily hiding the others. The show/hide button affects the visibility of elements only within the Edge workspace; it has no effect on the final project viewed with a browser.

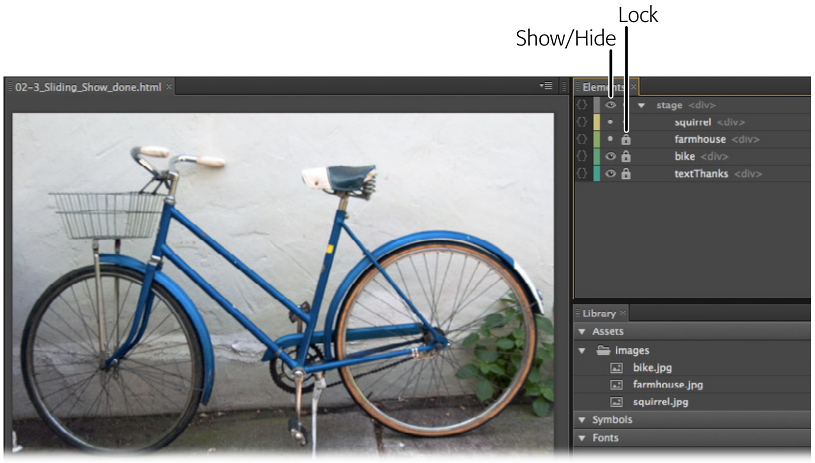


FIGURE 2-11

The Elements panel has two tools to help you when you're working with elements on the stage. The eye toggle button shows and hides elements on the stage. The padlock button locks elements into position to keep you from accidentally selecting and moving them.

When Edge imports artwork, it automatically puts the upper-left corner of the image in the upper-left corner of the stage. So the Location properties for imported images are X=0, Y=0. In the case of the squirrel, farmhouse, and bike, this is exactly where you want your images for the Sliding Show project described next. However, the images are 600 x 400 pixels, which makes them a little wider than the standard 550 px stage. That's easy enough to fix. Select the stage in the Elements panel and set W to 600 px.

The Sliding Show

Back in the olden days when photographs were recorded on film, there was an event called the *slideshow*. It was called a slideshow because the projector slid the images on and off the screen. These days, slideshows take the form of PowerPoint presentations, or they're used to display photos on a website. You can use a slideshow to show products. If you're selling cars, maybe you want to display all the color options. If you're selling home theater amplifiers, maybe you want customers to zero in on the different panels with controls and connectors.

In this next exercise, you create your own automated, sliding show. Each image is displayed for a period of time, and then it fades away as it slides offstage. You can use the three imported images from the previous exercise, or you can use as many of your own images as you want. Your show will look best if the slides are the same size and have the same portrait or landscape orientation.

1. Import your photos using File→Import.

   Edge places the images on the stage and automatically creates an *images* folder for your project. Copies of the photos are placed in the *images* folder. In the Library panel, you'll see the three images listed under Assets.

2. Select the stage in the Elements panel, and then set the stage dimensions and background color. Set the Overflow to hidden.

   Set the stage the same dimensions as your images. With overflow set to hidden, the images will disappear when they move off the stage.

3. Arrange your images so the first image is at the top of the Elements panel.

   In this project, the Z-order of the images determines when an image is displayed. It's as if you have a stack of snapshots on a table and you remove the top photo to see the next.

4. Shift-click to select all the images in the Elements panel, and then in Properties, click the Add Keyframe button next to Location, as shown in *Figure 2-12*.

   Translate (x) and Translate (y) keyframes are created for each of the images at 0:00 on the Timeline. This locks in the starting location for each of the images.

5. With all the images still selected, in Properties make sure the opacity is set to 100 percent and then click to add keyframes for Opacity.

   All the images will have an opacity of 100 percent until they begin to move off the stage.

6. Drag the playhead to 0:02, and then with the top image selected, add keyframes for Location and Opacity again.

   This locks in the location and opacity of the top image for the first 2 seconds of the slideshow. In the next steps, the photo will start to move and change. Make sure that the pin moves with the playhead to the new location. If necessary, press P to toggle the pin.

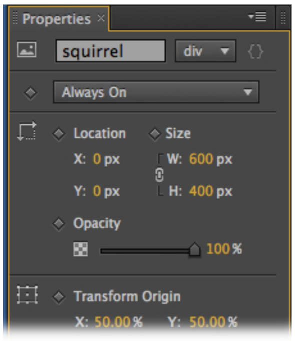


FIGURE 2-12

Click the diamond-shaped Add Keyframe buttons to create Location and Opacity keyframes in the Timeline. You can set the value for each property independently. Using the ID box, you can name the elements in your animation. This element is named "squirrel." Below the ID box is a menu that turns elements on or off, making them visible or invisible.

TIP If the Auto-Keyframe Properties button is clicked on, Edge will automatically create the Location property keyframes when you move the photo. If Auto-Keyframe Properties is not on, you can create keyframes by clicking the Add Keyframe button next to Location in the Properties panel.

7. Drag the playhead to 0:03, and then drag the top slide completely off the stage.

   You can have your slides move off the stage in any direction. Right movement imitates the old-fashioned slide projectors. Hold the Shift key as you drag if you want the image to slide along a vertical or horizontal axis.

8. With the image completely offstage, change the opacity to 20 percent.

   This creates a nice fading effect for each image as it moves offstage.

9. Drag the playhead down the Timeline another 2 seconds.

   This gives your audience time to admire your next photographic masterpiece.

10. Select the next slide that's visible on stage and then in Properties click the Add Keyframe buttons for Location and Opacity.

    This locks in the location and opacity for the slide that's showing. Like the previous slide, it will move and fade after this point.

11. Move the playhead another second down the Timeline and drag the photo offstage, and set the opacity to 20 percent.

    This completes the process for the second image in the slideshow.

12. Repeat steps 9–11 for each of the images in your slideshow.

13. When the last slide is removed from the stage, display a message to viewers.

    If you want, you can fade a message like "Thanks for watching!" on and off the stage to signal the end of the show.

You can preview your slideshow in Edge (*Figure 2-13*), or you can view it in your favorite web browser. You're probably already thinking of ways you can tweak and customize a slideshow like the one described in this exercise. Naturally you can change the timing for the appearance and disappearance of each slide. If you want a fancy pseudo-3D effect, you could use Skew and Size properties to make the slides look as if they are blowing away one by one. In future exercises (page 110), you'll see how to give your audience click controls for the slideshow.

On/Off: Another Way to Show and Hide Elements

So far this book has shown you a few ways to show and hide elements. For example, if you want to show and hide elements while you're working in Edge, you can use the visibility eyeball-icon in the Elements panel. That tool doesn't change the way your final animation looks, it merely hides elements while you're working in Edge. You've also seen how you can hide elements using their Opacity property. Set Opacity to 0 and presto-change-o—it disappears. Opacity is particularly good for animated effects like the one in the Sliding Show. Using Opacity, it's easy to make elements fade in and fade out. Still there's some work involved in controlling the opacity over the course of an entire Edge composition.

There's another, more fundamental way to show and hide elements on the stage: you can turn elements on and off. In the Properties panel, directly under the ID box (*Figure 2-12*), there's a drop-down menu that is initially set to Always On. So far, that's the option used in the exercises in this book. There are two other items on the menu: On and Off. Suppose you want to redo the Sliding Show so that images changed instantly without any motion or fading effects. That's a perfect job for the On/Off switch. Import your images as described on page 42. With all your images in place on the stage, set one image to On and set all the other images to Off. In the Timeline move the playhead to the two second mark (or any other interval that suits you). Turn the displayed image off and turn another image on. In this case the stacking order of the images (the Z-order) doesn't matter because only one image is visible at a time. If you want to review a working example of this method, check out *02-4_On_Off_Elements.zip* from the Missing CD at *www.missingmanuals.com/cds/edgepv5mm.*

There's one other thing to note, once you use the On/Off menu in Properties, Edge adds an On/Off toggle button in the timeline for that element. See *Figure 2-14*. This makes it quick and easy to set the playhead to a particular point in time and turn your elements on or off.

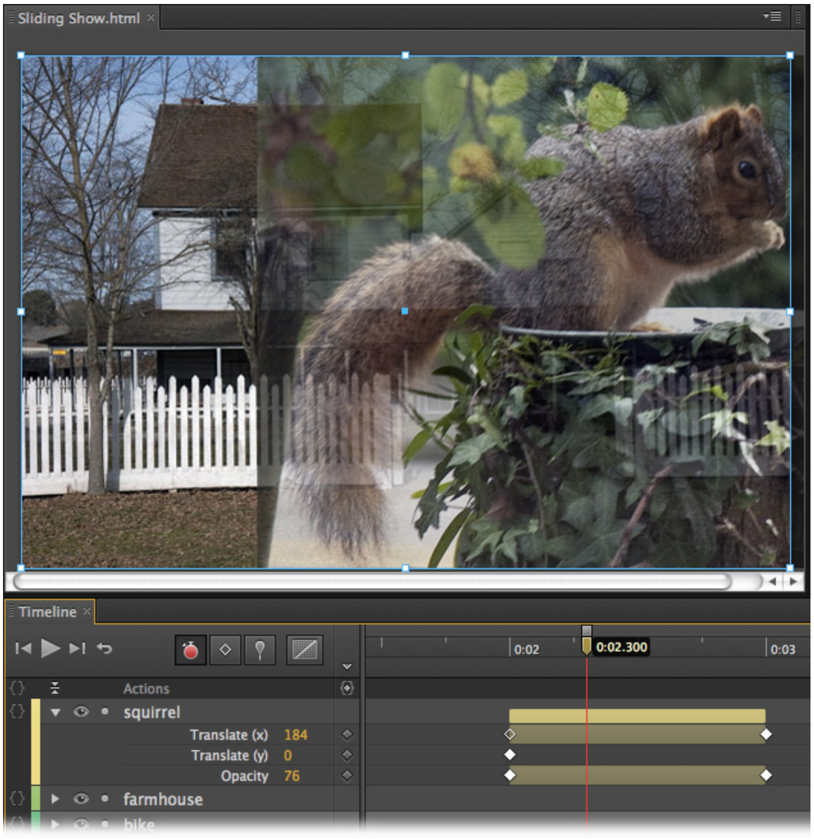


FIGURE 2-13

Here in mid-slide, the squirrel is leaving the stage and the farmhouse is revealed underneath. The movement and opacity of the slides is controlled by property keyframes in the Timeline.

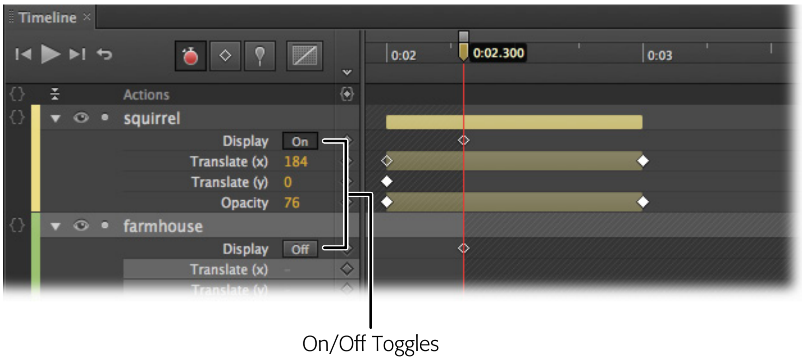


FIGURE 2-14

You can turn elements off until you need them and then turn them on. The initial control is a drop-down menu in the Elements panel. Once you turn an element on or off there, you'll see On/Off toggle buttons inthe Timeline.

CHAPTER

3

Adding and Formatting Text

In spite of the old saw about a picture being worth a thousand words, often words are the right tool for the job. When you want to label a button, build a menu-based navigation system, or provide how-to instructions for a particular task—it's time for text.

Edge is rooted in HTML5, so it gives you the same properties and text-handling features that you'd find in other web-building tools like Adobe Dreamweaver. That means you won't have all the typographic features that you'd find in a page layout program or a more complex graphics program like Flash. As you'll learn in this chapter, you do get your choice of fonts, and you can set the size, color, and alignment. If Edge doesn't have the special font that you need, you can use *web fonts*. For example, this chapter shows how to link to Google's web fonts. In addition, you'll learn how to apply transforms and effects just like those used with graphics and photos. Along the way, you'll learn how to animate your text, giving it a little bounce. Links are important to any web page. This chapter explains how to import and manage HTML text that includes links on specific words. HTML has a number of tags that are used to identify the content of text, such as block quotes from other sources and computer code used for examples. You'll learn which tags Edge uses and how to apply those tags to text.

Adding Text to Your Project

There are three ways to add text to your Edge project:

- **Use the text tool.** In the Tools palette, click the big T, and then in your document, click and drag to create a text box. Initially, you don't have to worry too much about positioning or sizing the text box. You can manage those details later. Just go ahead and start typing. Try the phrase "ON the EDGE." The text you enter appears on the stage, as shown in *Figure 3-1.*
- **Copy and paste.** If you're working with large blocks of text, you may have already worked up a draft in a word processor or some other source. In that case, you can copy the text in your word processor, and then in Edge create a text box and press Ctrl+V (⌘-V) to paste it into your project. It won't be formatted as it was originally, but the text will be there.
- **Open HTML with text.** Perhaps you have a web page already created in an HTML editor or some other web-building tool. You'd like to add some animation excitement to the static page. You can open that page in Edge using File→Open and then use Edge to animate the elements. Again, you lose the formatting, but this method will keep any links within the text.

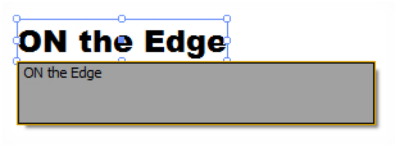


FIGURE 3-1

As you enter text in the lower box, it appears displayed on the stage in the upper box. The box with the blue border is the text box, which remains on stage when you're done typing. The lower box (sometimes called an IME for input method editor) disappears when you're through entering text.

About Text Containers

Once you've added text to your project, its location and size are managed by a text container called a text box. You can reposition your text by dragging it or by changing the Location properties. To move text manually, click the Selection tool (it looks like an arrow) or press the shortcut key V. As with other elements, the X and Y properties position the upper-left corner of the text box, so you can position text with precision by typing X and Y values in Properties. To reshape the text box, drag one of the handles or change the W or H properties. Keep in mind you need to delink the size properties to change W and H independently. Edge displays all your text whether it fits in the box or not, so if your text doesn't fit within the height of the box, it extends out the bottom.

Changing Text-Specific Properties

Once you have text in your Edge project, there are several text-specific properties that you can use to change its appearance, as shown in *Figure 3-2*.

- **Font name.** You can choose from several different typefaces. You might not find all the same fonts that you have on your computer. In web design, you're limited to fonts that are available to your audience unless you have a way of providing the font with your project. For more details on fonts and typefaces, see the next section. For a way to find and use additional fonts, see page 55.
- **Font size.** You dial in font size by number. Edge uses two different units for specifying font size: pixels (px) and ems (em). Pixels are equivalent to a single dot on a monitor. An em is roughly the size of the letter M. Most web browsers give users the ability to adjust the size of text, so an em is a unit that changes according to the browser setting.
- **Font style.** In addition to plain-vanilla text, Edge gives you three standard styles: bold, italic, and underline. You can apply more than one style to a font. Some of the styles may not be available for a particular font. For example, Arial Black is already a bold font, so the bold style has no effect.
- **Text color.** Choose a color for your text using the same color picker you use for rectangles and other graphics.
- **Align.** Just like your word processor, Edge lets you align text right, center, or left. Alignment affects all the text in the text box. So if you want to create one paragraph aligned right and one paragraph centered, they must be in separate text boxes.

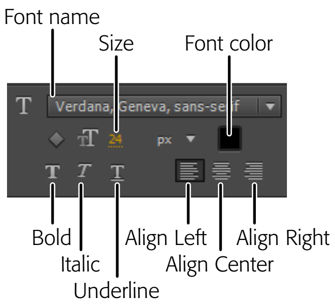


FIGURE 3-2

Edge provides the standard text formatting options, including choice of font, style, size, color, and alignment. In addition to these properties, you have access to the same properties available to graphics, such as size, location, rotate, and skew.

About Typefaces and Fonts

Choosing a typeface for your project should be fun—just not too much fun. Make your typeface decisions based on the job at hand, and you can't go wrong. Think about what you expect your type to do, and then help it do that job by choosing the right typeface, style, size, color, and alignment. Beginning designers often treat text as yet another design element and let the desire for a cool look override more practical concerns. Designers sometimes talk about a text block as if it's just another shape on the page. But cool type effects can torture your readers' eyes with hard-to-read backgrounds, weird letter spacing, or hopelessly small font sizes. (For more advice on readability, see the box on page 54.)

When you specify type for a web page, you have to take into consideration which fonts are available to your audience. If you specify a font that your audience doesn't have, the web browser will supply a substitute. Sometimes that substitute is close to your spec, and other times not so much. This has always been a challenge for web designers. In the past, designers could expect different fonts on Mac, Windows, and Linux computers. That situation has improved over the years, but most web designers always provide a list of two or three fonts, such as Arial, Helvetica, sans-serif. When a web browser sees the list, it uses Arial (originally a Windows font), if that's available. Next, it tries Helvetica—a font that was most common on Macs. If neither is available, the browser looks for some other sans-serif font. All of Edge's font specs use this multiple-choice method. In addition, the fonts listed are very likely to be available to your audience. Your choices are:

- **Verdana, Geneva, sans-serif.** Verdana (see *Figure 3-3*) was developed for Microsoft and is widely available on both Windows and Macs. The goal was to create a typeface that's readable on a computer screen in small sizes. Geneva was designed for the Apple with the same goal and was available on the original Mac. Both of these typefaces are good for long paragraphs of text. The term *sans serif* is borrowed from the French and means without serifs. Serifs in typography are the extra bars and strokes at the ends of letters that you see in some typefaces. In print, experts feel that serifs help make type more readable and lead the reader's eye along a horizontal line. Type isn't quite as sharp on computer screens, so the serifs' helpfulness isn't quite as obvious. (The text in this book is a sans-serif font.)
- **Arial Black, Gadget, sans-serif.** Arial Black and Gadget are both heavy fonts, meaning they have very thick lines, as shown in *Figure 3-3*. Use these typefaces when you're creating big, bold headlines.
- **Georgia, Times New Roman, Times, serif.** Times was a typeface developed for the New York Times newspaper way back when. It's a very readable typeface and works well for long paragraphs of text. Times Roman (*Figure 3-3*) and Georgia are similar to this much-imitated font.
- **Courier, Courier New, monospace.** Back in the old days, when office workers used typewriters, everything looked like Courier. The term *monospace* means that every character is provided the same space horizontally. This creates notice-

able gaps to accommodate narrow characters like "i," which get just as much space as "M". Monospace fonts are a little harder to read compared to a typeface like Times, but they do make a primitive kind of statement. Monospaced fonts are often used to display computer code.

- **Arial, Helvetica, sans-serif.** The monarchs of the sans-serif world, Arial and Helvetica, are used all over the place. They're also widely available to most computers. These typefaces can be used for body text or headlines. If you want a bolder typeface for a heading, choose Arial Black, Gadget, sans-serif.
- **Tahoma, Geneva, sans-serif.** Tahoma was designed for Microsoft and Geneva for Apple. Tahoma is a little narrower than its counterpart Verdana, which means you may be able to get more characters per line at a small sacrifice in terms of readability.
- **Trebuchet MS, Arial, Helvetica, sans-serif.** Trebuchet MS is a font that's more distinctive than Arial and Helvetica and not nearly as ubiquitous. A good typeface for headings of all sizes. It was also developed by Microsoft and widely distributed along with Office and other Microsoft applications.
- **Times New Roman, Times, serif.** The Times fonts are very versatile. Often used for body text, they can also be used for headings at large sizes. Times adds a sort of "old school" elegance to projects.
- **Palatino Linotype, Book Antiqua, Palatino, serif.** Palatino is a typeface that's been widely available on computers since the Mac first adopted it. An attractive and readable font, the Palatino family of typefaces has grown over the years. This specification refers to the original serif font.
- **Lucida Sans Unicode, Lucida Grande, sans-serif.** Lucida Sans Unicode was designed to supply a font that supported the most commonly used characters in the large Unicode standard. (The Unicode computer standard was developed to handle and display text used in most of the world's writing systems.)
- **MS Serif, New York, serif.** These are basic serif fonts provided by Microsoft and Apple, respectively.
- **Lucida Console.** Lucida Console and Monaco are monospaced, sans-serif typefaces, as shown in *Figure 3-3*. These are a good choice for menus or other user interface elements. In general, they won't work as well in large blocks of text.
- **Comic Sans MS, cursive.** These typefaces look more like handwriting than machine type (*Figure 3-3*). Great for special effects. You might strain your readers' eyes if you make them read too much with this spec.

Sometimes you may want to use a special or decorative font that's not included in Edge. Perhaps it's a company logo or a special heading style. In those cases, you might want to turn the text into a graphic. Create the text in a word processor or page layout program, and then take a screenshot to turn the text into a JPEG image. At that point, you can import the JPEG into Edge just like any other graphic element (see page 42).

NOTE If you want to be technical in a Gutenbergian fashion, typefaces are families of fonts. Times Roman is a typeface, while "Times Roman, bold, 12 point" is a font. Somewhere along the line, as type moved from traditional typesetters to computer desktops, the meaning of the word "font" came to be synonymous with "typeface." That's OK, but knowing how the terms originated makes great cocktail party banter.

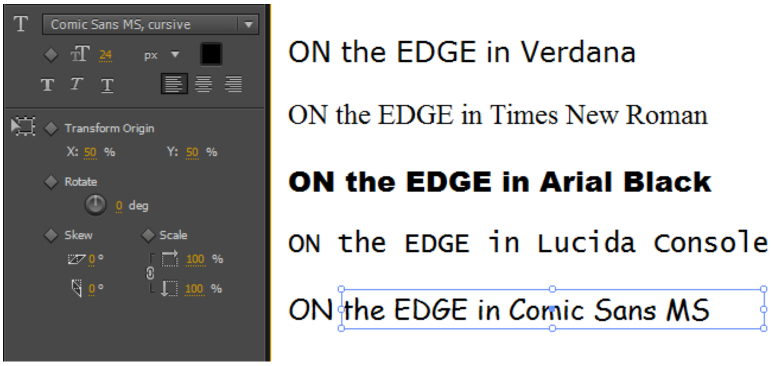

Comic Sans MS, cursive
24
px
Transform Origin
X: 50 %
Y: 50 %
Rotate
0 deg
Skew
Scale
0°
100 %
0°
100 %
ON the EDGE in Verdana
ON the EDGE in Times New Roman
ON the EDGE in Arial Black
ON the EDGE in Lucida Console
ON the EDGE in Comic Sans MS

FIGURE 3-3

Changing a typeface is as easy as selecting a font from the drop-down menu. However, it's important to choose the right typeface for the job. Arial Black is great for headings and bold statements, but it would be painful to read multiple paragraphs of the stuff. That job is better suited to Times New Roman or Verdana, which are very readable at small font sizes.

FREQUENTLY ASKED QUESTION

Small Is Beautiful

How can I use small type and make sure it stays readable? For most people, reading text on a computer or iPhone screen is more difficult than reading it off a piece of paper. If your Edge project includes text with small font sizes (12 px or less), there are a few things you can do to keep your audience from straining their eyes. Actually, the fact of the matter is, people simply won't read text if it's too hard to see.

- If possible, bump the type up to a larger size. At small sizes, a single pixel makes a big difference.
- Black text on a white background is easiest to read. If you don't want to use that combination, opt for very dark text on a very light background. If you have to use light text on a dark background, make sure there's a great deal of contrast between the colors.
- Use sans-serif type, like Verdana, Arial, or Tahoma for small sizes. Sans-serif type looks like the text in this book; it doesn't have the tiny end bars (serifs) you often find in type. Computer screens have a hard time creating sharp serif type at very small sizes.
- Use both upper- and lowercase type for anything other than a headline. Even though all-caps type looks bigger, it's actually less readable. The height differences in lowercase type make it more readable. Besides, too much uppercase type makes it look like you're shouting.
- Avoid bold and italic type. Bold and italic are often hard to read at small font sizes. It varies with different typefaces, so it doesn't hurt to experiment.

It never hurts to get second and third opinions. If you've got eyes like an eagle, you may want to get some opinions from your less-gifted colleagues when it comes to readability. You want your Edge project to be accessible to as wide an audience as possible.

Using Web Fonts

There's another way to increase the number of typefaces you use in your Edge animations. For years, web designers have been using *web fonts*. For programs, including web browsers, to display a specific font, they need to have access to the font description. Usually, that description resides on the same computer as the program—sometimes called the *client*. Web fonts work a little differently. For example, with Google's web fonts (*www.google.com/webfonts*), the definitions for the fonts are stored on Google's servers. As a web designer, you can use these fonts by adding code to your pages that tell browsers where to find the font descriptions.

NOTE In web-speak, the computer with the browser is called the client; the computer serving up the web pages is the host.

First, find the web font you want to use. Google web fonts are free and surprisingly easy to use, so they're a great candidate for your first attempt. Here are the steps to selecting a Google web font and grabbing the code you need to identify it in your project:

1. In your web browser, go to: *www.google.com/webfonts*.

   You see a page displaying font samples. There are hundreds, so the widgets on the left help you filter the fonts. The buttons at the bottom of the page direct you to the three steps for a successful web font hunt: Choose, Review, and Use.

2. On the left, below the word Filters, click the drop-down menu. Choose from Serif, Sans-Serif, Display, and Hand Writing.

   The menu uses checkboxes, so you can choose a combination of characteristics. For example, you could use Sans-Serif and Display.

3. If necessary, use the Thickness, Slant, and Width sliders to narrow your font search.

   With so many choices, it helps to thin the crowd of fonts displayed on the screen.

TIP At first, the Search box isn't much use to you because Google web fonts don't have the familiar names. After you use them a bit, you'll probably find a few favorites that you use repeatedly. If you remember their names, then you can use the Search box to quickly find that needle in the haystack.

4. Use the tabs at the top of the font window to change the display to Word, Sentence, or Paragraph.

   If you're looking for a font for headings, Word or Sentence, is the best choice. If you're choosing a font for body text, make sure you check its appearance with the Paragraph option.

5. Click the blue "Add to Collection" button.

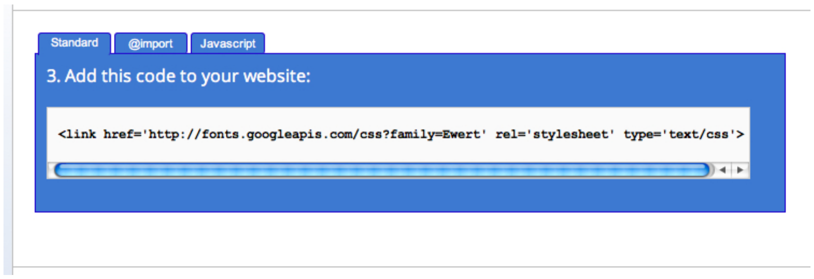
   You can have more than one font in a collection, but for page-rendering speed and good design, you'll want to limit the number of fonts you use.

6. Click Review.

   This step may not always be necessary, but as the name implies, on this page you can take a closer look at your font in use as a headline or paragraph.

7. Click Use.

   A new page loads with instructions for using the fonts on your web site. Part way down the page is a blue box with the heading "Add this Code to Your Website"; see *Figure 3-4*.



```
<link href='http://fonts.googleapis.com/css?family=Ewert' rel='stylesheet' type='text/css'>
```

FIGURE 3-4

Google provides several different code samples for using their web fonts on your website. The shortest and easiest one to use is under the Standard tab shown here.

8. Click the Standard tab and then select and copy the code displayed.

   With the code stored on your clipboard, you're ready and loaded for the second part of the process: adding the location for the font description to your Edge project.

Adding Web Fonts to Your Composition

Once you've chosen a Google or other brand web font and copied the code that identifies it, adding to your project easy in Edge. Here are the steps:

1. In the Library panel, on the bar that says Fonts, click the + button, as shown in *Figure 3-5*.

   The Add Web Font dialog box opens.

2. Paste the code that identifies the location of your font in the lower "embed code" text box.

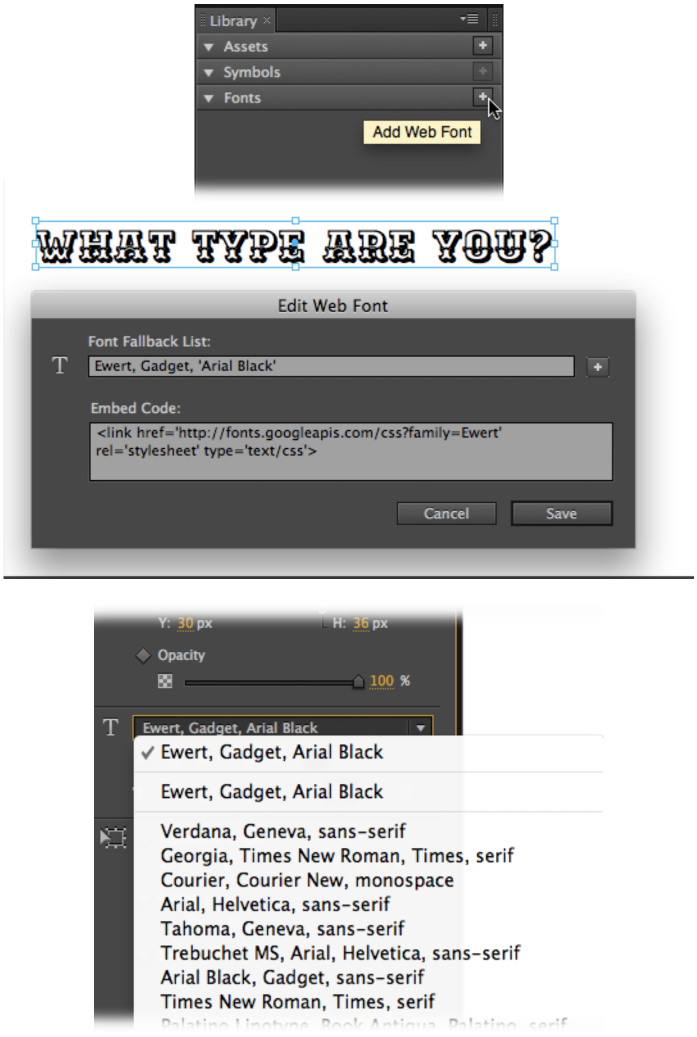
   This code is provided by the same organization that hosts the web font. If you followed the previous steps, the code is stored on your clipboard.

3. Type the name of web font in the upper Font Fallback List along with the fonts that should be used if the web font isn't available.

   If the client computer isn't connected to the Internet, then the web font won't be available.

4. Click the Add Font button.

   The font now appears in the Font Name drop-down menu when you're working with text.



Font Fallback List: Ewert, Gadget, 'Arial Black'

Embed Code:

```
<link href='http://fonts.googleapis.com/css?family=Ewert'
rel='stylesheet' type='text/css'>
```

Y: 30 px H: 36 px

100 %

Ewert, Gadget, Arial Black

✓ Ewert, Gadget, Arial Black

Ewert, Gadget, Arial Black

Verdana, Geneva, sans-serif
Georgia, Times New Roman, Times, serif
Courier, Courier New, monospace
Arial, Helvetica, sans-serif
Tahoma, Geneva, sans-serif
Trebuchet MS, Arial, Helvetica, sans-serif
Arial Black, Gadget, sans-serif
Times New Roman, Times, serif
Palatino Linotype, Book Antiqua, Palatino, serif

FIGURE 3-5

Three steps for adding fonts to your project. Top: Click the add font button. Middle: Paste in the code that identifies the location for the web font description. Bottom: Choose your new font from the Font Name box.

There are a few of things to keep in mind when you expand your font toolbox by using web fonts:

- It takes browsers a little longer to find, download and use web fonts compared to fonts stored on the local computer.
- Web fonts won't be available to computers that aren't connected to the Internet.
- Just because you have access to a bunch of new fonts, that doesn't mean you have to use them all. Don't sacrifice readability for novelty.

NOTE Google web fonts have the significant advantage of being free to use, but they aren't the only game in town. You'll find other options at *http://typekit.com*, *http://webfonts.fonts.com*, and *http://fontsquirrel.com*.

Changing Other Text Properties

Like any other element in Edge, you probably don't expect your text to be static all the time. Fortunately for you, the designer, you don't have to learn new tools to make your text dance around the screen. The X/Y Location properties determine where your text appears, and the W/H Size properties determine the dimensions of the text box. Keep in mind that the Size properties change the size of the text box, but they don't change the size of the letters. To change the size of the letters, you can use the Font Size properties or the Scale properties. As with drawings and photos, you can create property keyframes in the Timeline to make text properties change over time. For example, page 20 showed how to use the Opacity setting to make text fade in and out.

Remember those fold, spindle, and mutilate tools? You can use the Transform properties on text, too. Go ahead and rotate or skew blocks of text for special effects as you add or remove them from the web page. Use the Scale properties to make the text box and the text inside bigger or smaller. Scale works on text the same way it works on a JPEG image: Dial in a percentage, and everything grows or shrinks.

NOTE Edge anticipates how you're likely to animate your text. With this in mind, it automatically puts the transform origin in the middle of the text box when there's a single line of text. That makes it a cinch to create a headline that spins like a pinwheel. Naturally, you can reposition the Transform Origin at any time using the X/Y transform origin properties.

Clipping Text Around the Edges

As with other graphic elements on the stage, you can use Clip properties to hide the edges of a text box. It's a lot like cropping the edges of photograph. Suppose you want animate a text box so that at first only a pinpoint in the middle is visible, then it grows to display an entire block of text. At the bottom of the text properties panel, Edge provides four controls that represent the top, bottom, left, and right edges of the element (*Figure 3-6*, bottom). Type or scrub in values in pixels (px). As you make changes, you see the effect they have on your text box. Want to remove clip properties after you've applied them? Just right-click (Control-click) the clipped element and choose Remove Clip from the shortcut menu.

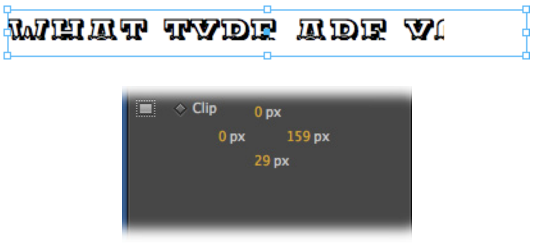

Clip 0 px
0 px 159 px
29 px

FIGURE 3-6

Clip properties give you an easy way to crop a text box. Top: The right and bottom edge of the text box are hidden. Bottom: That effect was created by increasing the values of the right and bottom clip.

Making That Headline Drop In

In most cases, the purpose of text is to communicate a message, so it's counter-productive to subject your audience to constantly moving and changing text. That doesn't mean you can't have a little bit of fun. For instance, you may want to have the heading on your web page drop down or bounce into place when the page first loads. Check out *03-01_Heading_Drop_done.zip* if you want to see an example.

In this project, you create a banner at the top of the stage. When the web page loads, three words—"ON the EDGE"—drop into place. In this case, you're animating the phrase "ON the EDGE." You break the words into three separate text boxes, so that you can move each word independently. In other cases, you may want to animate all the individual letters in a word or phrase. The toughest part of the trick is to get the letter or words to line up properly once they're in place. You want letter spacing to look natural, and you want the text to sit evenly on a horizontal line. Often, when you're animating words or letters like this, it helps to create a positioning template, and that's exactly what you do in this project. The positioning template (*Figure 3-7*) is visible at design time to help you align those moving words and letters. When you're done building the animation, you can remove the template.

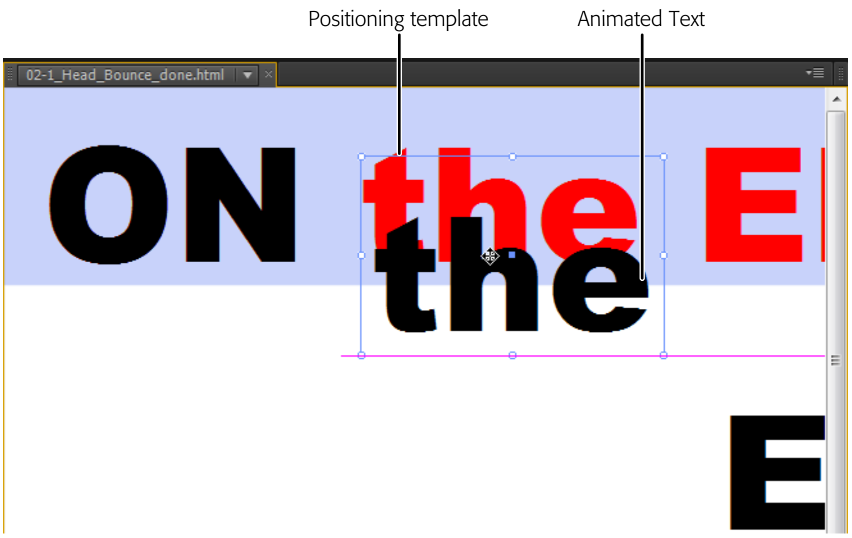


FIGURE 3-7

When you animate words and letters, it can be tough to get the spacing exactly right. If you create a positioning template, using a single text box, you can use it as a guide for placing words and letters in their final positions.

Here are the steps to create a drop-in heading:

1. Create and save a new 550 x 400 document with a white background color.

   As usual, create a new folder to hold the HTML and JavaScript files for your project.

2. With the Rectangle tool (M), create a rectangle 550 x 90 and place it at X=0, Y=0. Set the background color to R=200, G=210, B=250 and A=100%. Set border color to *none*. Name the rectangle *BannerBG*.

   The quickest way to make a rectangle to spec is to drag out a quick box that's any old shape and then type in the values in Properties. Make sure you click the link next to the W/H Size properties so you can enter nonproportional values.

3. In the Elements panel, click the Lock Element button next to *BannerBG*.

   A padlock appears next to *BannerBG*. Now, you can't accidentally select or move the blue box on the stage.

4. Select the text tool and drag out a text box. Then type *ON the EDGE*. Set the font to Arial Black; the size to 72 px; and the alignment to Centered.

   This text will serve as a positioning template for the animated text.

5. In Properties, give the text box the name *OnEdgeTemplate*.

   As with your graphics, you want to be able to identify different blocks of text in the Timeline and the Elements panel. At this point, the properties for the text look like *Figure 3-8*.

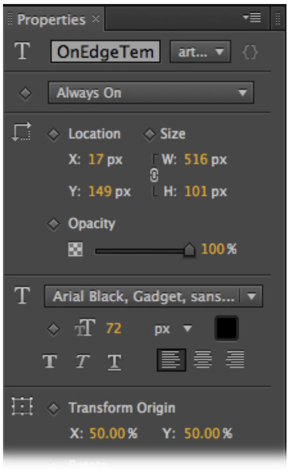
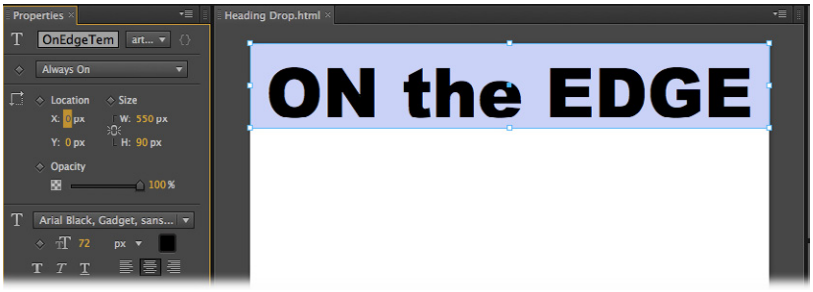

Properties
OnEdgeTem art...
Always On
Location X: 17 px Y: 149 px
Size W: 516 px H: 101 px
Opacity 100 %
Arial Black, Gadget, sans...
72 px
Transform Origin X: 50.00 % Y: 50.00 %

FIGURE 3-8

Arial Black is the thickest, boldest typeface in your text toolbox. It's well-suited to massive headings and titles. Using the 72 px font size and the Center Align property makes this banner fill the stage.

6. Set the text box's size and location to match the colored rectangle, with the size to 550 x 90 and the location to X=0, Y=0.

   When you're done, the top of the Edge stage should look like *Figure 3-9*. If for some reason the text is behind the blue box, you can change the Z-order in the Elements panel. Just drag *OnEdgeTemplate* so that it's above *BannerBG*.


Properties
OnEdgeTem art...
Always On
Location X: 0 px Y: 0 px
Size W: 550 px H: 90 px
Opacity 100 %
Arial Black, Gadget, sans...
72 px

Heading Drop.html

ON the EDGE

FIGURE 3-9

The blue box and your text box have identical Size and Location properties. That means the text appears centered over the blue box, making an attractive page banner.

7. Select *OnEdgeTemplate* and then press Ctrl+D (⌘-D).

   This duplicates the text, though you might not notice right away because it is placed right on top of the previous text. However, you can see *OnEdgeTemplateCopy* in the Elements panel.

8. Drag *OnEdgeTemplateCopy* down to the middle of the stage.

   In the next steps, you'll use this to create individual text boxes with separate words: "ON," "the," and "EDGE." Before that, it's a good idea to finish setting up the positioning template.

9. Select the original *OnEdgeTemplate* and then set the Text Color to red (#ff0000).

   As advertised, this text is being used for a positioning template. Later, the bright red color will make it easier to see if the text is correctly positioned.

10. In Elements, click the Lock Element button next to *OnEdgeTemplate*.

    This locks your positioning template in place so you can't accidentally select or move it.

11. Select *OnEdgeTemplateCopy* and press Ctrl+D (⌘-D) twice.

    This creates two more copies of the entire banner text.

12. Double-click the first *OnEdgeTemplateCopy*. In the text edit box, delete everything except the word "ON." Then in Properties, rename the text *ON*.

    It's best to eliminate extra spaces when you're animating single words or letters and you should reduce the width the text box to fit the edited text.

13. Repeat step 12 to make text elements for *the* and *EDGE*.

    You now have three properly labeled words that you can identify and animate independently. You may want to resize the text boxes' width to match the words, as shown in *Figure 3-10*.

14. Drag the word "ON" up so that it is above and slightly to the left of the stage. Drag the word "EDGE" so that it is above and slightly to the right of the stage. Drag the word *the* straight up so that it is above the stage.

    These are the starting positions for each of the words. They should be completely offstage.

15. Select "ON" and change the Rotate property to -30. Select "EDGE" and set the rotation to 30 degrees.

    These two words will appear to drop in at an angle from their respective sides.

16. Select all three words and click the Location and Rotate Add Keyframe buttons.

    The starting positions for each word are duly recorded in keyframes.

17. Drag the playhead to the half-second mark: 0:00.500.

The entire animation will take a second, which is plenty of time for a simple animation like this. You don't want to bore your audience. Each word will take a half-second to complete its move. Each word will start at a different moment.

TIP This animation uses fractions of a second in the Timeline, like 0:00.500 and 0:00.250. If you don't see those numbers on the Timeline, you can use the slider in the lower-right corner of the Timeline, as shown in *Figure 3-11*.

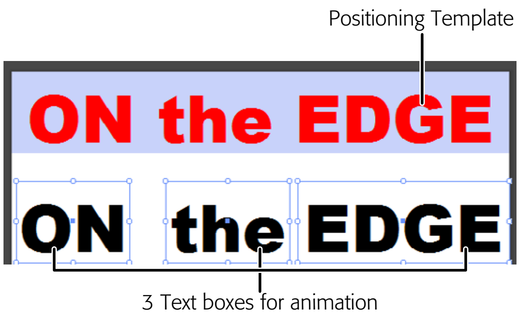


FIGURE 3-10

At this point, the text to be animated is temporarily placed in the middle of the stage. The red text over the blue banner serves as a positioning template. After it's served its design-time function, it will be removed.

18. Select "ON" and set the Rotate property back to 0, and move "ON" over the same word in the positioning template.

    If you want to review the motion, drag the playhead back and forth. If necessary, you can readjust the start or end point. Just move the playhead into position and tweak the word's position.

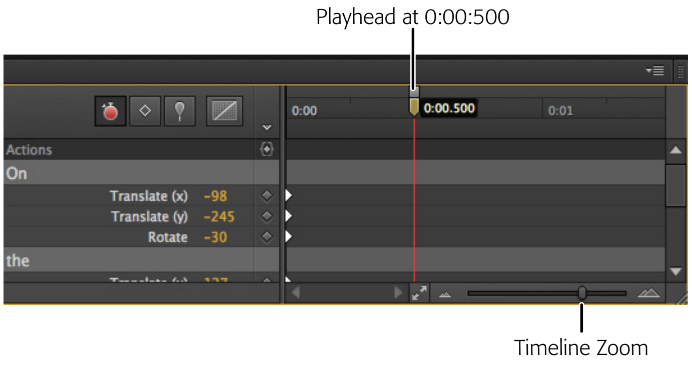


FIGURE 3-11

Use the Timeline Zoom slider to get a better view of the Timeline. The units of measure at the top of the Timeline change depending on the zoom level.

19. Move the playhead to 0:00.250. Then select the word "the" and click the Location Add Keyframe button.

    The plan here is to start the word "the" moving before "ON" has finished its movement. However, you want the word "the" to remain motionless for the first quarter-second, so you must create two location keyframes with identical values at 0:00.000 and 0:00.250.

20. Drag the playhead to 0:00.750, and then move "the" over the same word in the positioning template.

    No rotation is used for "the", so this word will appear to drop straight down.

21. Move the playhead to 0:00.500. Select "EDGE" and click the Location and Rotate Add Keyframe buttons.

    This keeps "EDGE" in place for the first half-second of the animation.

22. Move the playhead to 0:01, and then drag "EDGE" into place over the positioning template.

    At the 1-second mark on the Timeline, the words have finished their journey, and the first version of the animation is complete, except for removing the positioning template.

Before you remove the red positioning template, you probably want to preview the animation. Press Home and then the space bar to get a look. If necessary, you can continue to tweak the starting and ending points for the animated words. For example, you might prefer it if "ON" and "EDGE" drop in first and the word "the" is added last. If you'd like to add a little bounce to the words' entrance, see page 65.

Dealing with the Template

The red positioning template isn't meant to be a permanent part of the animation. So if you're happy with everything, you can remove it. First turn off the Lock Element button to make the Template selectable. Then you can select the template in either the Elements panel or on the stage and press Delete. As an alternative, you could turn the template into a drop shadow (*Figure 3-12*) or glow effect for the text.

- For a **drop shadow,** set the text to a mid-gray tone and then adjust the opacity to taste. Something around 30 percent usually works well. You might want to keep the drop shadow hidden until the three words have finished moving. If that's the case, set the opacity to zero until that point in the animation, and then bring it up.

- For a **glow effect,** choose a yellow or orange color. Use the Scale control to make the text slightly larger than the text that drops in place. Again, you'll probably want to use opacity to control the timing and appearance of the glow text. You might want the glow effect to fade in and then fade out, adding momentary emphasis on the heading.

NOTE The effects you create in this preview version of Edge aren't as slick as those you'd find in programs like Photoshop or Flash, and there aren't nearly as many options. If your project demands a more sophisticated presentation, you can always create graphics in one of those programs and then add them to your project. Just save images as PNG, SVG, GIF, or JPEG files.

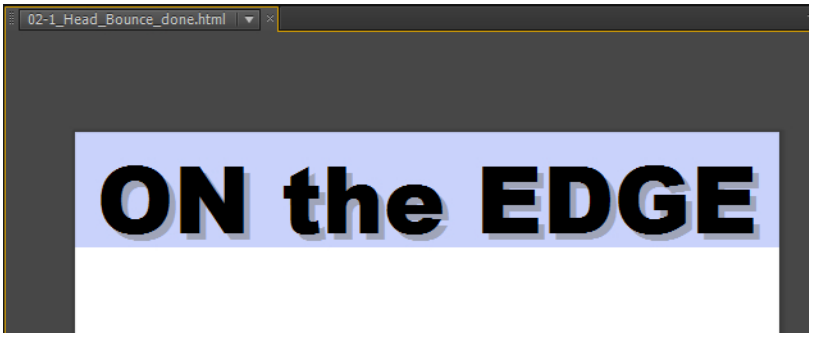


FIGURE 3-12

You can turn the positioning template into a quick and easy drop shadow for your drop-in heading. Apply a gray text color and some opacity and you're ready to go.

Adding Some Bounce

If the previous example, where text drops into place, is too sedate for your web page, you might want to consider adding a little bounce to the action. Bounce makes it seem like your web page adheres to the laws of physics. Like a basketball, your text can start with a big bounce and then one or two smaller bounces until it settles into place. You can create your own bounce by adding position keyframes, or you can create a bounce using the Easing properties that are part of the transition in the Timeline. Chapter 4 covers the Timeline and all of its features in much greater detail. This section will quickly cover some bounce basics.

Creating a Bounce Manually

You can take a crack at creating a bounce manually using the *03-2_Word_Bounce .html* project. Open the file and examine the Elements panel—you'll see the stage with three other elements. "BOUNCE" is the word that you'll animate. "BounceTemplate" (red text) is the positioning template. As in the previous example, this marks the final position for the animated text. The ground element is a gray rectangle that's positioned at the bottom of the stage. You can think of this as the ground on which the text will bounce.

A bouncing motion is created in the Timeline by adding keyframes with alternating up and down locations (*Figure 3-13*). With the playhead still at 0:00, select BOUNCE and click the "Add Keyframe for X and Y" button in Properties. This sets the starting point. For the next leg of the journey, drag the playhead to 1:00 and move BOUNCE so that it covers BounceTemplate. Click "Add Keyframe for X and Y" to add new location keyframes. Drag the playhead to 1.75 and then move BOUNCE up near the middle of the stage. Move the playhead to 2.25 and then move BOUNCE back over the template. You can create a few more bounces using a shorter period for the motion—half a second, then a quarter second. With each bounce up, shorten the distance.

When you're done, your project will look something like *03-3_Word_Bounce_done .zip*. When you get tired of a straight up-and-down bounce, you can always add a little rotation to the movement, making it look like the word is bouncing back and forth off of the lower corners. If you reduce the vertical scale property when the text hits the ground, you can create a cartoon-like smooshing action, as if the text were compressing on impact with the ground.

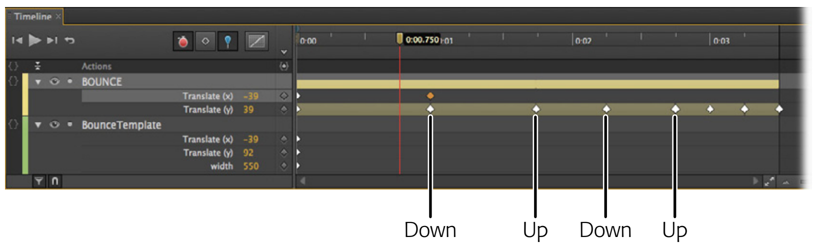


FIGURE 3-13

You can make just about anything bounce in Edge, including text boxes. It's simply a matter of creating position keyframes that alternate an element's position. Here the up-down motion is created using the text box's Y property.

Using Edge's Prebuilt Bounce

First, a little background about transitions and the concept of easing. When you animate an element on the stage, by changing properties and creating keyframe properties in the Timeline, you create transitions. Those transitions are shown visually as bars in the Timeline. Like the elements on the stage, transitions have properties, too. One of the properties is called Easing. In the real world, when objects move, they accelerate and decelerate. You never see a car begin to move at full speed or come to a stop instantly. The Easing properties help you create more realistic movement by automatically controlling an element's transition. It just so happens that one of the easing options helps you to create a bouncing motion.

Here are some steps to explore transition properties:

1. Open *03-4_Easing_Bounce.html* in Edge.

   The stage looks suspiciously similar to the Word Bounce exercise.

2. Drag the playhead to 0:01, and then drag BOUNCE down so that it covers BounceTemplate.

   Edge creates a transition in the Timeline.

3. In the Timeline, click the transition lane next to BOUNCE.

   The transition in the Timeline is highlighted.

4. At the top of the Timeline, click the Easing button.

   The easing panel appears above the Timeline. Initially, the tooltip for this button says Easing: Linear, because that is the easing method that's applied. With linear easing, the transition is applied at a steady rate from beginning to end.

5. On left side of the Easing panel, click Ease Out. Then, on the right, click Bounce, as shown in *Figure 3-14*.

   When you click Ease out, the panel displays a number of Ease Out methods. The graph gives you a visual representation of the easing method.

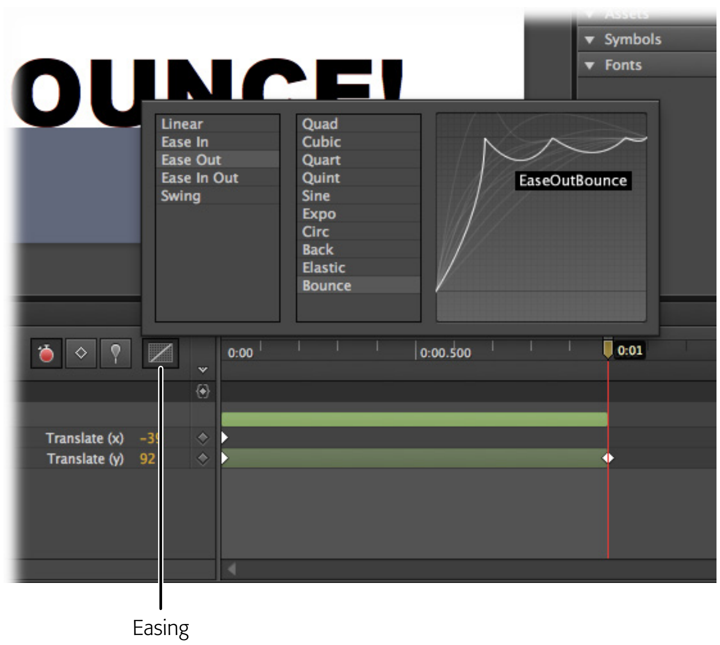


FIGURE 3-14

Edge provides a number of different easing options for transitions: EaseOut-Bounce makes an element bounce at the end of the transition. If you want the bounce at the beginning, choose EaseInBounce.

6. Click outside of the Easing panel.

   The panel closes and your easing method is applied to the selected transition.

7. Press Home and then the space bar.

   When the animation plays, you'll notice some nice bouncy action at the end of the motion. If you'd applied EaseInBounce, the bouncing motion would have occured at the beginning of the transition.

The easing properties for transitions can be a real timesaver. Edge includes a number of different transitions, but sometimes the names are a little cryptic. The best way to learn the different easing characteristics is to create a practice animation and apply different eases to identical elements and transitions.

Adding Links to Text

As you'll see in Chapter 5, you can add links to the elements you place on the stage using Edge's triggers and actions. Often you'll want to have a link on one or two words within your text. The simple way to handle that is to create your web page in your favorite web builder and then save it as an HTML file. Then, using Edge you can open that HTML file. Once the HTML text is in Edge, you can animate the text box just as if you'd created it in Edge.

In this exercise you can use the file *03-5_Outside_Text.html* from the Missing CD (*www.missingmanuals.com/cds/edgepv5mm*), or you can use one of your own files. The missing CD file is just about the simplest HTML document that you could create, which makes it easier to find the line of text once you've imported it into Edge. The code looks like this:

```
<!DOCTYPE html>
<html>
<head>
</head>
 <body>
    <p>You can learn everything in a <a href=
    "http://www.missingmanual.com"> Missing Manual.</a></p>
   </body>
</html>
```

You'll learn more details about HTML in upcoming chapters. At this point it's helpful to know that HTML code uses pairs of tags. Tags are identifiable by their angle brackets: <tag_name>. For example, in this code there is a paragraph of text that says, "You can learn everything in a Missing Manual." The paragraph is placed inside two tags. The <p> tag marks the beginning and the </p> marks the end of the paragraph. Inside the paragraph is a hyperlink, which also uses a pair of tags, <a> and </a>. When you're specifying a link, you need to provide an address for the link. That is done within the first tag, so it reads like this: <a href="http://www.missingmanual.com">. The *href* stands for hyperlink reference. It's followed by the assignment operator (=) and then the address is placed within quotes.

1. While you're in Edge, choose File→Open and open the 03-5_Outside_Text.html document.

   When the file is open, the Elements panel lists two items matching the tags in the document: <p> and <a>. Note that the <a> tag is inside the <p> tag, just as the hyperlink is inside the paragraph. Click the triangle button to expand and collapse the <p> element in the Elements panel. You can check to make sure the link works by going to File→Preview In Browser.

2. Select <p> in the Elements panel.

   A text box appears around the text on the stage, and the properties are displayed in the Properties.

3. Create a simple drop-and-bounce animation for the text, using the techniques described on page 59. Use the Transformation tool to double the size of the paragraph. Skew the text slightly at the beginning if you want it to appear to fly in from an angle.

   You can animate and change any of the properties for the <p> element just as you'd animate other graphic elements inside Edge.

4. Go to File→Preview In Browser and click the hyperlink.

   The text dances around the page, and through the whole process the hyperlink is active and clickable. If you're connected to the Internet, it takes you to the Missing Manuals website.

As shown in the example, you can change the properties for the elements you loaded from an HTML file. The one thing you can't do is edit the text in the <p> or the <a> elements from within Edge. If you need to fix typos or make other changes, you can do that by editing the HTML file in a text editor like Notepad++ (Windows) or BBedit (Mac). Just make sure you save the file as HTML.

TIP You can add Edge text (and other elements) to an HTML document that you open in Edge. So, for example, you might have body text that was originally in the document, with hyperlinks throughout. Then, you can create some animated headings using Edge's text tool.

HTML Tags in Edge

As shown in the previous example, Edge speaks HTML and makes use of some of the standard HTML tags. You can apply tags to the text you create in Edge, as shown in *Figure 3-15*. One of the major advantages of doing this is to help you format text consistently. For example, all your paragraphs may be displayed in a serif font at a font size of 12 px. Your major heading <H1> might use the 24 px font size, while the next heading <H2> uses 18 px.

All of these tags come in pairs, and the closing tag includes the slash character, like so: <div>This is the content inside.</div>

- **<div>** The div tag is used to define a section within a document. You can think of it as dividing the document into parts. The div tag is particularly popular in Edge because you use it to define any element you want. Once the element is defined with div, you can make it move.

- **<address>** The address tag is used to identify the author of the document and to provide contact details.

- **<article>** The article tag is new in HTML5. It is used to specify content that may be from another source.
- **<blockquote>** The blockquote tag is used to define a long quotation.
- **<p>** The paragraph tag has long been one of the most common tags in HTML.
- **<h1>** through **<h6>** These tags define headings, each of which can have distinctive formatting.
- **<pre>** The pre tag designates preformatted text. Usually HTML ignores extra white space; however when text is within the pre tags, it is displayed with its original formatting.
- **<code>** The code tag defines computer code, making it available for special formatting such as monospaced fonts.

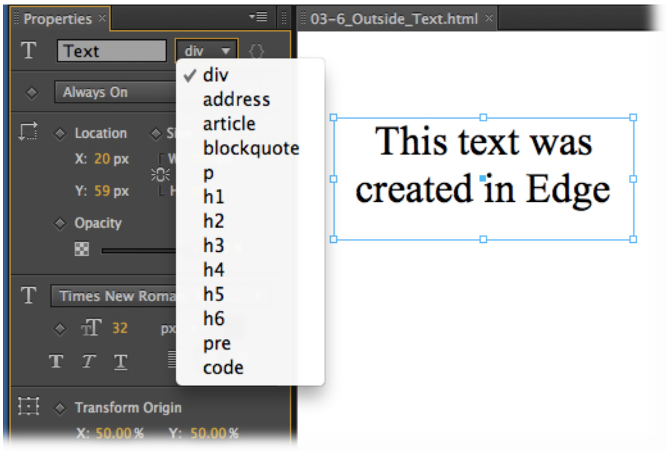


FIGURE 3-15

When you select a text box in Edge, you have the option of applying common HTML tags to the text. In the Properties panel, click the drop-down menu next to the element name.

PART

2

Animation with Edge

CHAPTER

4

Learning Timeline and Transition Techniques

The art of animation is all about images changing over the course of time in a natural, pleasing, and entertaining manner. It's the same whether you're creating a cartoon with a long-eared rabbit or you're developing a presentation for the next quarterly sales meeting. Elements move, change shape, and change color. In Edge, that means that the properties that define an element change over the course of time. Those properties and their changes are tracked on the Timeline through the use of *keyframes*—those little diamond-shaped markers.

The previous chapters involved some Timeline manipulation. This chapter provides more complete details on Timeline basics and controls. You'll learn how to create Timeline labels and how to set, move, and remove keyframes. And of course you'll explore transitions, learning how to tweak them to do your bidding. When you're through, you'll know how to operate every button and widget the Timeline has to offer.

Introducing the Timeline

Master the Timeline, and you'll be an Edge Jedi. You'll have a jump on the learning process if you've used a timeline in a video editor, Adobe After Effects, or Flash. If you tackled any of the exercises in the earlier chapters, you're not a complete stranger to the Timeline. When you work in Edge, you use three panels to create your animation: Elements, Properties, and the Timeline. Usually you jump back and forth among them, using their features as necessary. It's no surprise that the Timeline is the panel that's devoted to working with time—selecting specific moments in your animation and making something happen.

When you first look at the Timeline, it seems to be quite complicated with all its buttons and widgets. Create a couple of transitions, and things don't get any simpler down there in Edge south. Don't be intimidated. Think of the Timeline simply as a ruler that measures time in your animation. See *Figure 4-1*. The playhead lets you select a certain moment in time. For example, drag it to 0:02 on the Timeline, and the stage displays the elements as they appear 2 seconds into your animation. As explained in the earlier chapters, elements' position on the stage and their appearance is controlled by properties: Location properties, Color properties, Size properties, and so on.

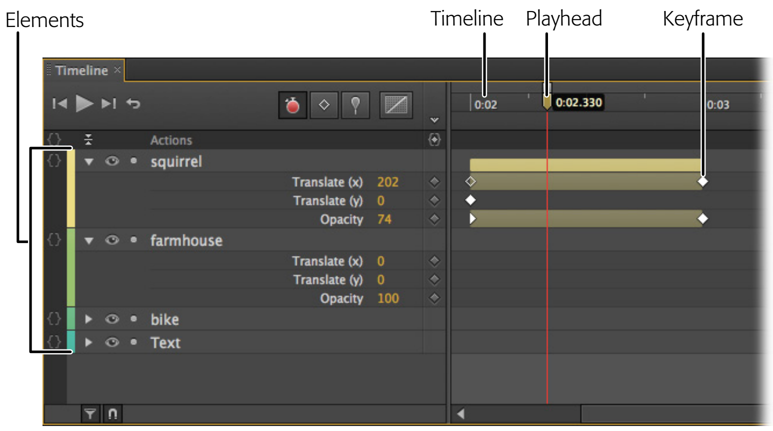


FIGURE 4-1

The Timeline lists elements in your animation and their properties. The playhead lets you select a certain moment during the animation. Keyframes mark a point in time when the value of a particular property changes.

Initially, the Timeline displays numbers like 0:01 and 0:02, with minutes on the left side of the colon and seconds on the right. As you're working, you'll want to zero in on a particular portion of the Timeline and the property keyframes it holds. You can zoom in and out of the Timeline; however, doing so changes the tick marks and numbers displayed. The controls for zooming in and out of the Timeline are in the lower-right corner, as shown in *Figure 4-2*. When you zoom in on the Timeline, you'll begin to see fractions of seconds displayed, like 0:00.500 and 0:00.250, as shown in *Figure 4-3*.

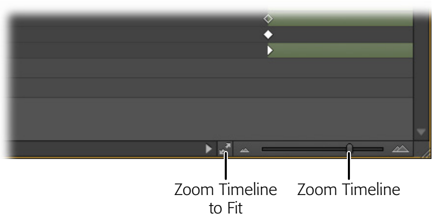


FIGURE 4-2

Want a better view of a particular segment of the Timeline? Drag the slider to zoom in and out of the Timeline. Click the Show All button to see the entire active portion of the Timeline.

Choosing a Moment in Time

To select a particular moment in time, drag the playhead along the Timeline. The playhead and its red marker line selects a point in the animation's run time. As you move the playhead, the time counter to the right of the playhead changes to display the selected time in numbers (*Figure 4-3*). The three playback buttons next to the time counter work just like the ones on your iPod or Blu-ray player. The left button jumps to the beginning of your animation, and the right button jumps to the end, wherever that may be on the Timeline. The big triangle in the middle plays your animation in real time. If the animation is playing, the same button works as a pause button. You won't always want to play your entire animation. Sometimes, you're just interested in a segment. The play button starts to play your animation from the playhead's position. After you've watched your animation, click the return arrow next to the playback controls to move the playhead back to its previous position.

TIP If you can see the point in time you want to select, but the playhead is missing in action, click the Timeline. The playhead will jump to that spot.

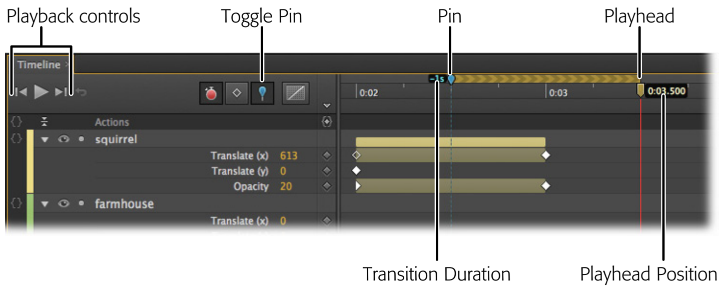


FIGURE 4-3

On the far left of the Timeline, you have playback controls, just like those on your iPod. The time counter numerically displays the position of the top playhead. You use the playhead and the pin to mark two points in time when you create transitions.

As you drag the playhead around, you may notice some snapping action as the playhead jumps to a particular tick mark on the Timeline's ruler or jumps to a keyframe. That snapping action is deliberate, but you're in complete control over how it works. Go to Timeline→Snapping to turn the feature on or off, as shown in *Figure 4-4*. (That's right. The Timeline is important enough to have an entire menu of its own.) To fine-tune snapping behavior, you can choose the things that the playhead snaps to. Go to Timeline→Snap To and then turn on or off a snapping feature. These are your choices:

- **Quarter Seconds.** The playhead snaps to tick marks in the Timeline in quarter-second increments, like 0:00.250, 0:00.500, 0:00.750, and 0:01.
- **Playhead.** The playhead comes in two parts, top and bottom. With this feature turned on, the two parts snap to each other when they're moved. For more details on using the bottom portion of the playhead while making transitions, see the note on page 83.

- **Transition Edges.** Transitions are displayed in the Timeline as horizontal, colored bars. With this option on, the playhead snaps to the keyframes at the beginning and end of transitions. Often you'll want to change multiple properties at a single point of time. If you turn on Snap To→Transition Edges, it's easier to target those points in time.

The Snap To options work when you're moving the playhead, but that's not the limit of their assistance. Later on, as you edit and fine-tune animations, you'll find snapping handy for lining up keyframes and the edges of transitions.

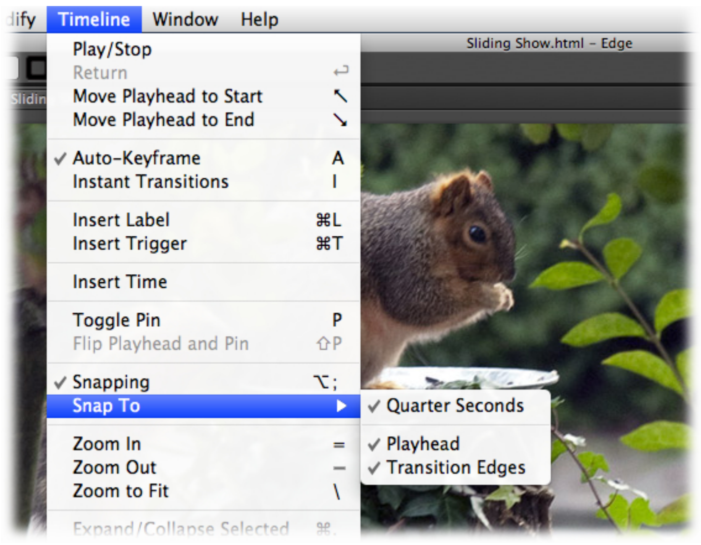


FIGURE 4-4

With snapping on, the playhead jumps to specific things in the Timeline, and you don't have to be such a mouse marksman. With snapping off, you're in complete control to move the playhead as you wish.

TIP In some cases, your Timeline may be too long to fit on the screen. If that's the case, you see a horizontal scroll bar at the bottom of the Timeline. You can use this just like you use scrollbars in a web browser to change your point of view.

Adding Labels to the Timeline

Computers like numbers more than most human beings do. People appreciate the precision of numbers, but often the descriptive value of a word like *BOUNCE* or *ChangeSlide* trumps something like 0:03.720. You can add labels to the Timeline to mark important points in your animation. They appear right beneath the number scale, giving you the benefit of both a word and a number.

Drag the playhead to that special point on the Timeline that you want to label. Then go to Timeline→Insert Label (or press Ctrl+L or ⌘-L). A label appears on the Timeline with a less-than-helpful name like Label 1 or Label 2. See *Figure 4-5*. It's up to you to give the label a useful name. Just start typing after you create a label.

When you're done, that label serves as a meaningful bookmark for you in the future. What's more, as you'll see on page 107, labels are extremely useful when it comes to creating triggers and actions that control your animation.

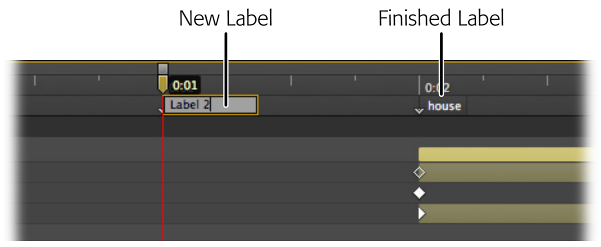


FIGURE 4-5

Use Timeline labels to mark the important points in your Timeline. As you'll see on page 107, you can also use labels in JavaScript when you want to jump to a point in the Timeline or to trigger an event.

Editing Labels

Labels aren't chiseled in stone once you create them. Change the name of a label by double-clicking on the label and typing in a new name. Unlike elements, labels can include the space character, so they can be more than one word. However, on the Timeline, labels display only about six or seven characters, so it's best to keep them short and sweet.

You can move labels to a different point on the Timeline by clicking and dragging them to a new spot, as shown in *Figure 4-6*. To delete a label, click to select it. Once it's highlighted, press Delete or Backspace (just Delete on a Mac).

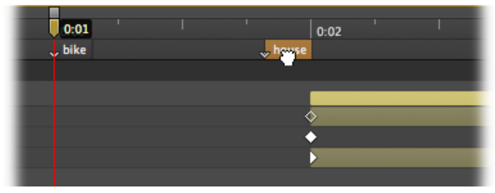


FIGURE 4-6

Click to select a label. When it's highlighted you can delete it or move it, as shown here. If you're moving a label along the Timeline, don't worry about the other labels. They won't bump into one another. Simply drag the selected label past any others on the Timeline.

Understanding Elements' Timeline Controls

The busiest part of the Timeline is over on the left side, where the elements are listed. Elements are the objects displayed on the stage. Actually, even the stage itself is considered an element, as you can see in *Figure 4-7*. As explained back on page 26, each element has a number of properties that you use to control it. For example, text has a Color and a Size property. Rectangles have Width and Height properties. All visible objects, except the stage, have Location properties: X and Y. These properties are listed in in "lanes" under the element's name in outline fashion, with properties subordinate to their element. Click the triangle button to show or hide the property lanes under an element. You can also use the Timeline→Expand/Collapse Selected option as long as you have an element selected in the Timeline, or its related shortcut key Ctrl+period (⌘-period for the Mac). There's also a handy command that expands and collapses all the elements at once. Use Timeline→Expand/Collapse All. The shortcut for that command is Ctrl+comma (Shift-⌘-comma on the Mac).

TIP To the left of each element name, there's a small color bar. Edge automatically labels each element with a specific color that you see in the Timeline and the Elements panel. The same color appears on the transition bars in the Timeline. This color coding of elements makes it a little easier to quickly identify them. The colors don't serve any other function.

Even though you're building an animation, that doesn't mean every single element in your animation is going to change over time. Perhaps you have several background elements that remain static through the whole show. In that case, there's no need to clutter up the Timeline with elements that aren't animated. Click the Only Show Animated Elements button (*Figure 4-7*) to filter out the lazier elements in your animation. When necessary, you can always toggle them back on.

Next to each element in the Timeline, you'll see, grayed out, the HTML tag that defines it. In many cases, that tag will be <div>, the generic tag that can be used in HTML to define any object, no matter how big or small. You may also see image tags <img>, paragraph tags <p>, or any number of other common HTML tags. The same tags appear next to the element's name in the Elements panel. In the Properties panel, you see a drop-down menu next to the element's name. You can use that menu to change the tag applied to an element.

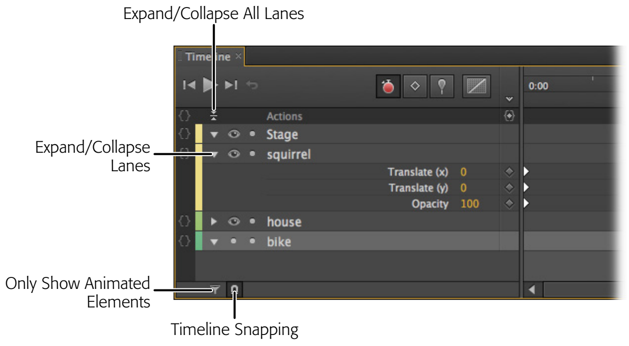


FIGURE 4-7

Each element in your project, including the stage, gets a row in the Timeline. Click the Expand/Collapse button next to the element's name to show or hide the properties that are used in your animation. Here the squirrel properties are hidden, while the farmhouse properties are shown. Bike doesn't show an Expand/Collapse button because no properties have changed.

Showing, Hiding, and Locking Elements

When you're working with a lot of elements and a complex animation, you tend to have some of those "can't see the forest for the trees" moments. You need to hide some elements on the stage so you can focus on others. That's the job of the show/hide toggle buttons next to each of the element's names in the Timeline. See *Figure 4-8*. When the button looks like an eyeball, the element is visible. When it looks like a period, the element is hidden. Keep in mind that this affects elements' visibility only while you're working in Edge. It doesn't affect your finished animation when it's viewed in a web browser. To control the visibility in the finished animation, you need to use the On/Off property, the Alpha property, or the JavaScript show/hide methods, as explained on page 173.

Accidentally moving an element while you're working is another design-time problem. Perhaps you want to select some text but accidentally grab a background photo, and now you've dragged it out of position. To avoid these problems, you can lock elements in place. When they're locked, you can't move them—you can't even select them in the Elements panel or on the stage. The Lock button looks like a lock when it's enforcing the no-move, no-select rules. When an element is unlocked, the button looks like a period (*Figure 4-8*).

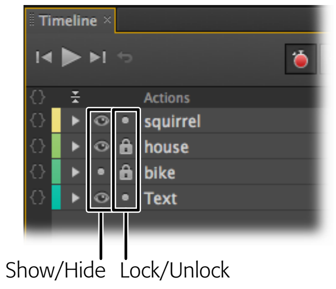


FIGURE 4-8

The elements part of the Timeline is very busy with buttons and toggles. Many of these controls duplicate controls elsewhere. For example, the show/hide and lock/unlock buttons work exactly as they do in the main Elements panel.

Using Timeline Keyboard Shortcuts

Like most applications, Edge gives you several ways to do the same thing. When it comes to managing the Timeline, you may be able to perform the necessary acts using the menus, buttons, or keyboard shortcuts. The good thing about keyboard shortcuts is that they give you a quick way to trigger a command, such as "create a label" or "turn on Auto-Keyframe Properties." The bad thing about keyboard shortcuts is that you have to remember the key combinations. That means you'll probably learn just a few shortcuts for the things you do the most.

Here's a quick cheat list for common Timeline tasks, along with their menu commands and keyboard shortcuts:

ACTION	MENU COMMAND	WINDOWS KEYBOARD SHORTCUT	MAC KEYBOARD SHORTCUT
Play or Pause the animation	Timeline→Play/Pause	Space bar	Space bar
Go to the beginning	Timeline→Move Playhead to Start	Home	Home
Go to the end	Timeline→Move Playhead to End	End	End
Zoom in on Timeline	Timeline→Zoom In	=	=
Zoom out on Timeline	Timeline→Zoom Out	–	–
Show all active time	Timeline→Zoom to Fit	\	\
Toggle snapping on or off	Timeline→Snapping	Alt+;	Option-;
Add a label to the Timeline	Timeline→Insert Label	Ctrl+L	⌘-L
Add trigger to Timeline	Timeline→Insert Trigger	Ctrl-T	⌘-T
Toggle Auto-Keyframe Properties	Timeline→Auto-Keyframe Properties	A	A
Toggle Instant Transitions	Timeline→Generate Smooth Transitions	I	I
Expand or collapse a selected element's properties	Timeline→Expand/Collapse Selected	Ctrl+. (period)	⌘-. (period)
Expand or collapse all elements' properties	Timeline→Expand/Collapse All	Ctrl+Shift+. (period)	Shift-⌘-. (period)
Activate the pin portion of the playhead	Timeline→Toggle Pin	P	P

Creating Transitions

Transitions have been part of this book since the first exercise in Chapter 1, so chances are you've created a transition or two by now. This section examines transitions and highlights some of the ways you can tweak and modify them. If you want to experiment along with the discussion, go ahead and create and save a new Edge document. Turn off Auto-Keyframe and the other buttons on the top of the timeline. Draw a rounded rectangle that's about 100 to 140 px—no need to be too precise. Drag the playhead away from the starting point 0:00. Now, move the element. As expected with Auto-Keyframe Properties turned off, no keyframe or transition is created. This is simply the new location for the element along the whole Timeline. Scrub the playhead a bit, and you can confirm this.

Creating Instant Transitions

A traffic light changes from red to green. A headline says "98 Degrees in the Shade" then flashes "It's HOT!" Those are instant transitions; there's no gradual change over the course of time. Here's a quick experiment you can perform with the Missing CD project called *04-1_Instant_Transition* (*www.missingmanuals.com/cds/edge-pv5mm*). A simple traffic light graphic is made from a black rectangle and three colored circles. The elements are appropriately named: redLight, yellowLight and greenLight. Initially the opacity for redLight is set to 100% while the opacity for the other two lights is set to 30%, making them appear turned off.

1. At 0:00 on the timeline click the Add Keyframe button next to Opacity for redLight, yellowLight, and greenLight.

   You can select all three lights in the Elements panel and then click Add Keyframe for Opacity once. The advantage of selecting elements in the Elements panel is that you won't inadvertently move them on the stage. After you click the button, Opacity lanes appear in the timeline for each of the elements. As shown in *Figure 4-9*. Next to the word Opacity, you see the value for the property and a second Add Keyframe button you can use to create new keyframes for that specific property.

2. In the Timeline, turn on Auto-Keyframe (A) and Instant Transitions (I).

   These buttons work as advertised, Auto-Keyframe automatically creates keyframes when you change a property and Instant Transitions, prevents Edge from creating a gradual transition for the property.

3. Move the playhead along the timeline a second or two. Then change the Opacity value for redLight to 30% and the value for yellowLight to 100%.

   You can change the Opacity property for each element in the Properties panel, or you can input the value in the property lane. When you change the value, new keyframes automatically appear in the timeline.

Test your animation and you see the instant transition as one light turns off and the other turns on.

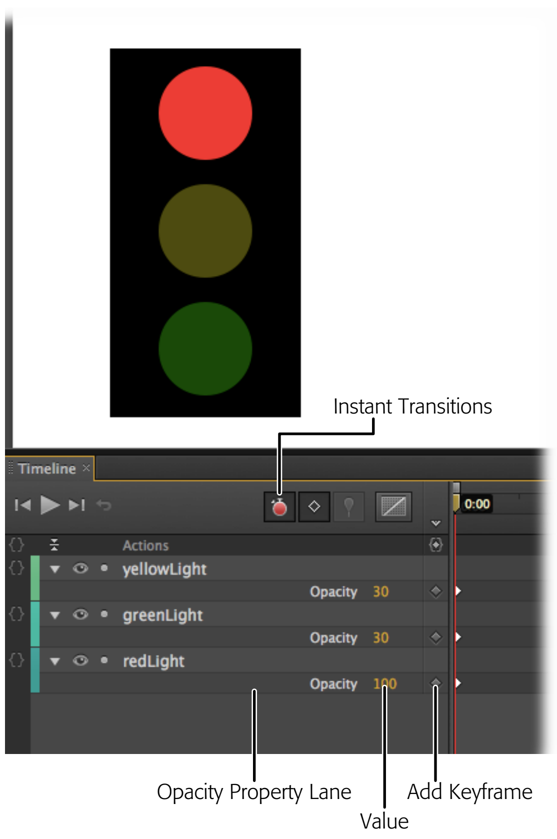


FIGURE 4-9

Turn on the Instant Transitions button when you want a property to change from one value to another without a gradual transition. You can change the property values and add keyframes using the controls in the property lanes of the timeline.

Setting the Pin for a Smooth Transition

When you want to animate Newton's apple falling from a tree, Pinocchio's nose growing, or a car driving down the road, you need to create a smooth or gradual transition. Often you'll want to make these kinds of smooth transitions as the value of a particular property changes. Smooth transitions aren't limited to a location on the stage. You can smoothly change from one color to another—the element will pass through several shades of the two colors. You can smoothly change the size of an element, making it appear to grow gradually.

Here's a quick experiment you can do with a rounded rectangle or any other element:

1. Create a new Edge project and turn on Auto-Keyframe (A) and turn off Instant Transitions (I).

   When Auto-Keyframe Properties is turned on, Edge automatically creates keyframes when you change an element's properties.

2. Draw a rectangle, rounded rectangle, or some other element on the stage.

   Any old element will work for this quick experiment.

3. Turn on the Pin (P).

   When the pin is toggled on, you can move the playhead and the pin separately. This makes it easy to mark two positions on the timeline at once. More importantly, it makes it easy to mark the beginning and end point for a smooth transition.

4. Drag the playhead to 0:01 and move the pin to 0:00.500.

   A gold transition bar appears in the timeline with arrow pointing toward the playhead. The blue numbers next to the pin display the length of time for the transition.

5. Move the element to a new location on the stage.

   Edge creates keyframes, one at a 0:00.500 marking the original location of the element and one at 0:01, marking the new location, as shown in *Figure 4-10*. Colored bars signal a smooth transition, and if you scrub the playhead over the transition, you'll be able to appreciate the smooth motion.

NOTE It's worth taking a moment to consider how the two-part playhead works. Both the top and the bottom portion create keyframes. The changes you make on the stage are recorded in the keyframes at the top playhead position. The mark (bottom part) creates keyframes that match the property values before the change. Why are these keyframes that duplicate property values important? In the previous example, if you don't set the mark, the element's motion would begin immediately at 0:00. Setting the bottom playhead ensures that the rounded rectangle remains motionless between 0:00 and 0:00.500. The element's motion begins at the point of the mark, and it ends at playhead position 0:01.

In the Timeline, there are numbers to the right of the Translate (x) and Translate (y) properties. These values are editable by typing or scrubbing. Notice that at the beginning of the transition, the value for each property is 0—that is, the element's original location. At the end of the transition, the numbers show how far the element moved from that original position. Negative numbers indicate leftward or upward movement.

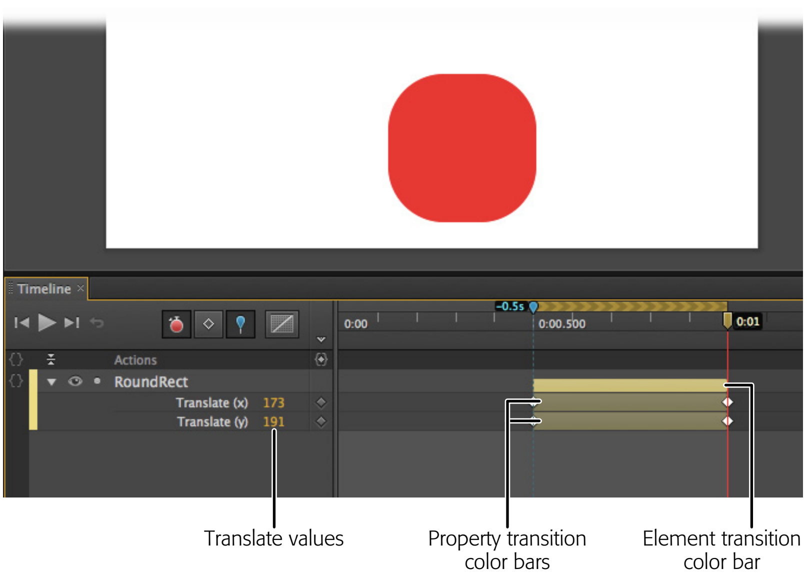


FIGURE 4-10

Color bars mark smooth transitions in the Timeline. The colors match the identifying color of the element. You can select a single property transition by clicking on the bar. You can select all the transitions by clicking the element's bar.

Your transition has three colored bars: the two bars for the properties and one bar at the top for the element. For example, it might be RoundRect, as shown in *Figure 4-10*. If you collapse RoundRect to hide the properties, you still see the top bar as a signal that there's a transition.

It's worth noting that the pin doesn't always have to be positioned behind the playhead in the timeline. So, for example, in the current animation you can put the playhead at 0:01.500 mark and the pin at 0:20 and create another transition, by moving the rectangle again. Keep in mind the element's new location is marked by the playhead while the pin retains the value prior to the move. The result is the rectangle moves to the new location (playhead) and then returns to the previous position (pin). This is a handy trick whenever you want to create a repeating or yo-yo kind of motion or if you want to an element to grow and then shrink back to its original shape.

Manually Adding and Removing Property Keyframes

Sometimes, you'll want to add keyframes manually. Exercises earlier in the book showed how to create keyframes with the Auto-Keyframe (A) button toggled off. Simply click the Add Keyframe button next to the property you want to add (*Figure 4-11*). To create a transition, you usually need to have two keyframes with different values—see the box below. If you want to remove a property keyframe, the process is similar to deleting a word in a document. Click to select the doomed keyframe. It shows a gold highlight. At that point, just press Delete (Mac or PC) or Backspace (PC). If you delete the two and only keyframes for your rounded rectangle, it still remains visible on the stage. You've deleted the keyframes that recorded its position, but you didn't delete the element.

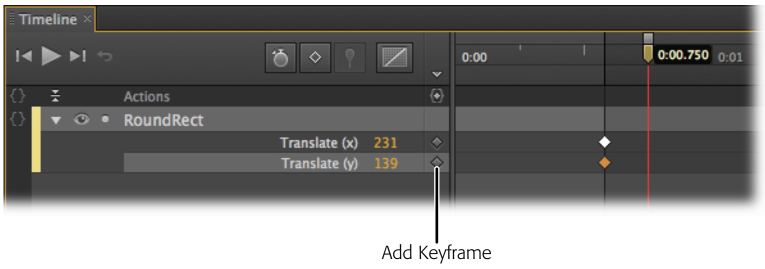


FIGURE 4-11

Here the Translate (x) and Translate (y) properties each have a single keyframe in the Timeline. As a result, there won't be a visible change. It takes two property keyframes with different values to make a change in the Timeline.

UP TO SPEED

Don't Forget the Starting Position Keyframe

Consider the following scenario: You've got Auto-Keyframe Properties turned on. You import a photo to your project. Edge automatically places the images on the stage with the upper-left corner at X=0, Y=0. You move the playhead down to the 2-second mark and then move the photo to another location. That's fine and good. But when you move the playhead back to 0:00, the photo is no longer in its original position: X=0, Y=0. What happened?

In short, you're missing a keyframe. Even with Auto-Keyframe Properties turned on, Edge doesn't create a keyframe when it imports a graphic. It creates keyframes only when properties change. When you're starting out with Edge, the previous scenario and similar gotchas occur every so often. Just keep in mind that when transitions or changes don't work as expected, it's very possible you're missing a keyframe at the beginning of the animation. It takes at least two keyframes for a transition or a change in an element to take place—one at the beginning and one at the end.

So the solution is to explicitly create a keyframe for the starting position of the element. If you remember to do this when you add elements to the stage, you'll avoid these problems.

Editing Transitions

In most animations, timing is critical. If you're animating a bouncing basketball, your audience will immediately notice if the ball hangs too long in midair. If you're syncing a slideshow to a music track, your audience will notice if you miss the beat. So don't be surprised if you don't get it right the first time you create a transition. Often, it'll need a little tweaking to get the timing just right. Fortunately, it's not hard to tweak transitions in Edge, and there are a few different ways to do it.

You can change the timing of transitions by dragging individual keyframes or the entire transition up and down the Timeline. For example, if you drag the keyframe on the end of a transition to the right, making the transition bar longer (as shown in *Figure 4-12*), you extend the time it takes for the transition to take place. If you're animating a car and you extend the transition, the car drives slower, taking more time to reach its destination. Want the car to go faster? Make that transition shorter.

When it comes to changing the duration of a transition, you don't have to edit each property separately. If you want to change all the properties at once, select the element bar. This selects all the properties below. Then you can drag the ends to change the duration.

In some cases, you'll want to move a transition without changing its duration. That's easy to do, too. Just select the transition you want to move and then move the cursor over the middle, until it changes to a hand. Then click and drag the transition to its new home.

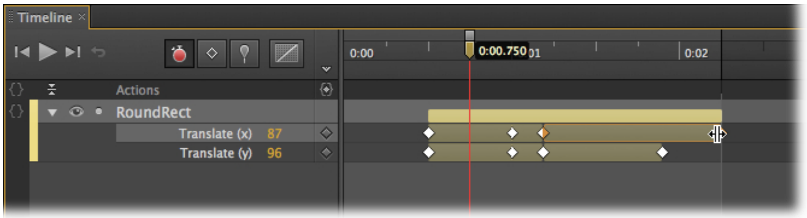


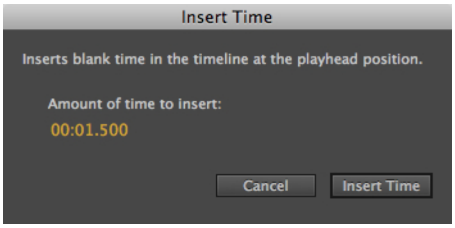
FIGURE 4-12

You can change the duration of a transition by dragging the keyframe at the beginning or the end. Here, the keyframe at the end of the Translate (x) property is being dragged down the Timeline, extending the transition. RoundRect will continue to move horizontally after the vertical motion stops.

TIP The Timeline displays your element's properties and the transitions visually. Often that's just the ticket for fine-tuning a particular effect. If you need numerical precision, you can use the playhead to check the time on the beginning and end of the transition.

The Insert Time command can also be used to extend the length of a transition. Move the playhead to the point where you want to add more time, and then choose

Timeline→Insert Time. As shown in *Figure 4-13*, a dialog box appears, where you can type or scrub in a time value in the usual 0:00.000 format. Bear in mind that the Insert Time command affects the entire Timeline. If three transitions are taking place at that point in time, Insert Time extends them all. It also adds to your project's overall running time.


Insert Time

Inserts blank time in the timeline at the playhead position.

Amount of time to insert:
00:01.500

FIGURE 4-13

You can insert time at any point in the timeline by moving the playhead to a position and then using the Timeline→Insert Time command. Dial up the amount of time you want to add and click the Insert Time button shown here.

Adding Easing, Reversing Motion, and Fine-Tuning Transitions

Want to practice fine-tuning a transition? Here's a project that will give you a little exercise. You can find *04-3_Basketball_Bounce* with the Missing CD folders at *www.missingmanuals.com/cds/edgepv5*. The goal is to make the basketball bounce with a realistic motion. As the ball bounces on the big letter B, it squashes the letter against the ground. When the ball comes back up, the letter re-inflates to its natural stature. See *Figure 4-14*. Along the way, you'll apply some easing to the basketball bounce, and copy and paste a transition. Most likely, you'll want to adjust the timing a bit by shortening or lengthening the transition bars.

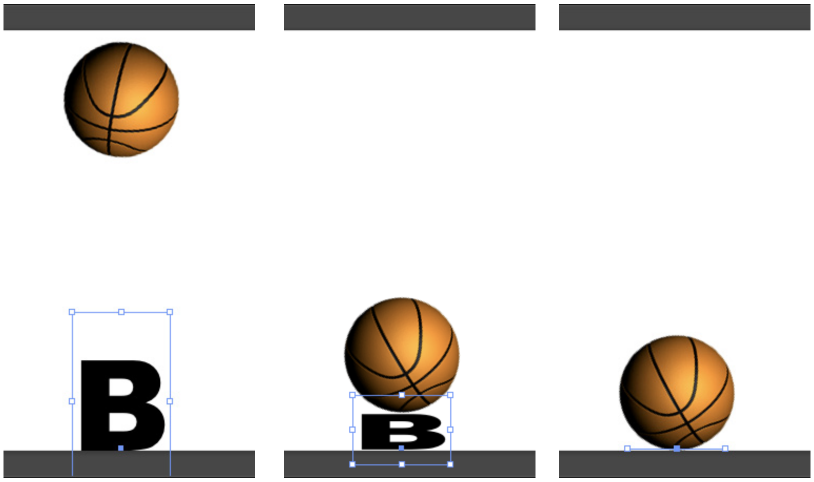


FIGURE 4-14

In this animation, the basketball drops from above (left). When it meets the letter B, it squashes the letter (middle). By the time it hits the ground, the letter is flat (bottom). As the ball bounces up, the letter springs back into shape.

Here goes! Follow the bouncing ball.

1. Open *04-3_Basketball_Bounce.edge* in Edge.

   There are two elements on the stage: a basketball image and a text box with a big bold letter B sitting at the bottom of the stage.

2. Press *A* to turn on Auto-Keyframe Properties, and make sure that Instant Transitions (I) is turned off and then toggle the pin (P).

   The shortcut keys give you a quick and easy way to toggle these features on and off. As an alternative, you can click the buttons on the Timeline or use Timeline menu commands.

3. Drag the playhead to 0:01 and leave the pin at 0:00.

   With the playhead and pin in this position, you can create a transition that lasts 1 second—a good first try for the basketball's downward motion.

4. Drag the basketball down so that its bottom edge is at the bottom of the stage.

   The basketball should hide the letter B, for the most part. The bottom of the basketball graphic should snap to the bottom of the stage. If you drag the ball straight down, only one property is added to the Timeline, the basketball's Translate (y) property.

5. Click to select the basketball's Translate (y) and press Ctrl+C (⌘-C).

   This copies the transition. You can now paste it back into the Timeline. You can paste it into a different Timeline location, or you can paste the transition in the Timeline for a different element.

6. With the playhead still at the end of the transition, go to Edit→Paste Special→Paste Inverted.

   This handy command not only pastes the transition into the Timeline, but it also reverses the action. So the basketball will go from the bottom of the stage back to its original position. You now have both motions for the bounce.

7. Press Home and the space bar to test the animation.

   The ball goes up and down, but you wouldn't call it realistic animation. It feels a bit wooden. Time for some easing.

8. Select the first section of the Translate (y) transition. Then, in the Timeline, click the Easing button. When the Easing panel appears, choose *Ease in* and then choose *Cubic*.

   When you first see the transition panel, Easing is set to linear. That makes the ball travel at the same speed the whole length of the motion. With the *Ease In, Cubic* applied, the ball moves more slowly at the beginning of the journey, picks up speed, and is moving faster by the time it hits the ground. If you're curious, go ahead and test some of the other easing options.

9. Select the second section of the Translate (y) transition and apply the *Ease Out, Cubic* easing.

   This easing option is the reverse of the first one. That means the ball will slow as it reaches its highest point on the stage.

10. Press Home and the space bar to test the animation.

    The motion feels better. The ball seems to react to hitting the ground, and with the slowing down at the top, it feels like gravity is in play.

11. Move the playhead to the 0:01 point where the ball hits the ground. Then select the ball and set its Rotate property to 40 degrees.

    The ball rotates rather lazily as it drops. Realistic enough. You can give it more or less spin by changing the Rotate value.

12. Move the playhead to 0:02 and set Rotate to 80 degrees.

    This makes the ball continue to spin in the same direction. If you prefer, use a negative number to make the ball spin back when it hits the ground. At this point, the ball is doing pretty well. Time to focus on that letter B.

13. Drag the playhead to the point where the ball touches the top of the letter B, then select the letter B, as shown in *Figure 4-15*.

    The Properties panel shows B's properties, including Transform Origin and Scale. The transform origin is the point around which transformations such as rotate and scale take place. For more details, see page 31. When B is selected, the transform origin is shown as a blue square. It starts out centered in the text box.

14. Adjust the Transform Origin Y property so that the transform origin is at the bottom of the letter B.

    With the transform origin at the bottom of the character, that point will stay in place when you apply transformations like Scale.

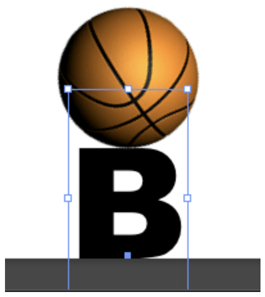


FIGURE 4-15

Here the transform origin (blue square) for the text box is positioned at the bottom of the stage. Now the bottom of the letter will stay positioned at the bottom of the stage when transformations are applied.

15. In Properties, with B selected, click the Add Keyframe button next to Scale.

    Edge creates a keyframe where B is scaled at 100 percent.

16. Drag the playhead to the point where the basketball hits the bottom of the stage. Then set B's vertical scale to 0 percent.

    This makes it look like the basketball is squashing the letter.

17. Select the letter B's Scale (y) transition bar and press Ctrl+C (⌘-C).

    The transition is copied, and now you can paste it back into the Timeline at a different point.

18. With the playhead at the end of the transition, and B still selected, go to Edit→Paste Special→Paste Inverted.

    As with the basketball motion (step 6), this reverses the scale transition. So, as the basketball bounces up, the letter bounces back to its regular size.

At this point, the basketball bounce is pretty good. If you want to compare your work to another example, check out *04-4_Basketball_Bounce_done*. Naturally, there's a lot more you can do. For example, if you don't find the motion realistic enough, you can speed it up or slow it down by adjusting the length of the transitions. To adjust everything at once, Shift-click to select both the basketball element and the B element. Then drag the end of the basketball transition. Don't like the results? A simple Undo (Ctrl+Z or ⌘-Z) puts everything back the way it was. If you're ambitious, you can have the basketball bounce out an entire word. (LeBron, anyone?) Or you can apply the bounce-squish to some of your own artwork. Experimentation is an artist's best tool.

Dealing with Timeline Claustrophobia

If you're working on a computer with modest screen real estate, the area available to the timeline is probably minimal. That becomes a problem when your projects get bigger and more complex. The best long-term solution is to move up to a larger monitor or add a second monitor. Many computers, including laptops, let you run two monitors at once. With that kind of a setup, you can devote an entire monitor to your timeline. That gives you room to see all of the elements and their properties at a glance. It will definitely cut down on that constant scrolling to find a new moment on the timeline. If your time is valuable a large monitor or a second monitor will pay for itself.

An alternative short-term solution is to become friends with the Maximize Frame command shown in *Figure 4-16*. In the upper-right corner of the timeline, there's a drop-down menu. The last item on that menu is Maximize Frame and it works as advertised. Choose that option and the timeline takes over the Edge workspace, hiding the other panels and the stage. With the Timeline maximized, you can tweak your transitions in comfort. When your work is done, go to the same menu and choose Restore Frame Size. Voila, you're back at the old, cramped layout.

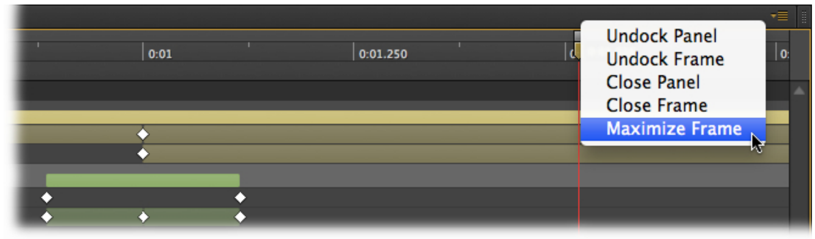


FIGURE 4-16

When the timeline seems too small for serious work, use the Maximize Frame command shown here. After using the command, the timeline will fill the available space hiding the other panels, but giving your room for serious timeline manipulation.

CHAPTER

5

Triggering Actions

The great beauty of the World Wide Web is that it's interactive. Unlike a static newspaper or magazine, with a mouse click or a finger swipe, you can make something happen. Perhaps you're flicking your way to the next photo in a slideshow. Or maybe you're jumping to view a new segment of an animation or movie. When you're building websites, the best way to hold your audience's attention is to put them in control of their experience. Let them turn features on and off. Let them quickly access the content that's most important to them.

This chapter shows you how to give your audience interactive control over their web experience. As a developer, Edge gives you tools called triggers and actions. You get to choose which events act as triggers. It might be a mouse click, or it might be the playhead reaching the end of the Timeline. Then you can specify the actions that take place. For example, you can change the size, color, or transparency of a clicked element, or you can jump to a new point in the Timeline. As always, Edge translates your project into JavaScript/jQuery code, but it makes development easy for you. In some cases, adding a trigger and an action is as easy as choosing items from a menu. In other cases, you'll need to tweak the code a bit to make it work according to your needs. This chapter walks you through the process. You'll learn how to use triggers and actions to do things like showing and hiding elements on the stage. You'll see that the Timeline has some special features when it comes to triggers. Near the end of the chapter, you'll revisit the slideshow project from Chapter 2, learning how to make it interactive.

Elements, Triggers, and Actions

If you're building interactive web pages, you'll love triggers and actions. The very words "trigger" and "action" describe how they work. Some event takes place (the trigger), and that makes something happen (the action). Triggers are attached to elements like the stage, the Timeline, a rounded rectangle, a car graphic, or a box of text. That means you have different triggers for different elements, as shown in *Figure 5-1.* A mouse click, for example, is an obvious trigger for an element on the stage. However, you can also create a trigger in the Timeline. For example, when the playhead reaches the end of the timeline (complete), you can specify an action like going to another point in the Timeline to start playing again ("Play from").

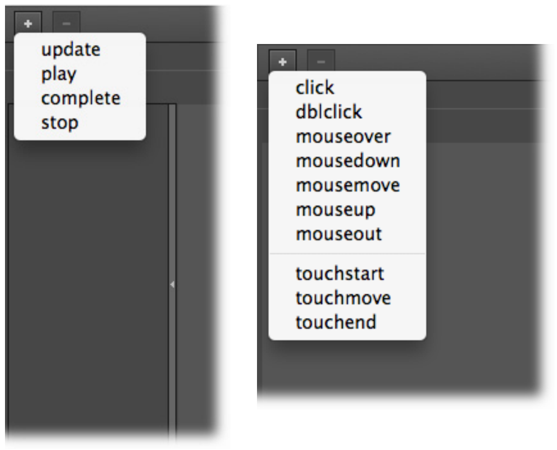


FIGURE 5-1

As you can see here, different elements have different types of triggers. The Timeline has triggers like play, complete, and stop. A graphic like a rectangle has triggers like click, mouseover, and touchmove.

Building this Edge-style interaction or automation involves three major steps:

1. Choose an **element**.
2. Set the **trigger**.
3. Specify the **action**.

Notice the icons that look like curly brackets in Properties, Elements, and the Timeline (*Figure 5-2*). Edge calls these Open Actions buttons, and by design, they are already attached to specific elements. See how each element in the Elements panel has Open Actions buttons? The Timeline, even though it's not included in the Elements panel, has its own Open Actions button. Click one of those buttons, and a panel opens with the name of the element in the panel's title bar. You're immediately prompted to choose a trigger, like *click* or *dblclick*. Once you've done that, you're two-thirds of the way home. The last thing you need to do is specify the action, which you can do by choosing from the buttons on the right side of the Actions panel. These pre-built actions pop the necessary code into your project and display it in the panel, where you can fine-tune it when necessary.

TIP At a glance you can tell whether an element has any triggers and actions attached. If the element has none, the brackets are empty and appear grayed out. If there are triggers and actions attached to an element, the brackets are bright white and there's block inside.

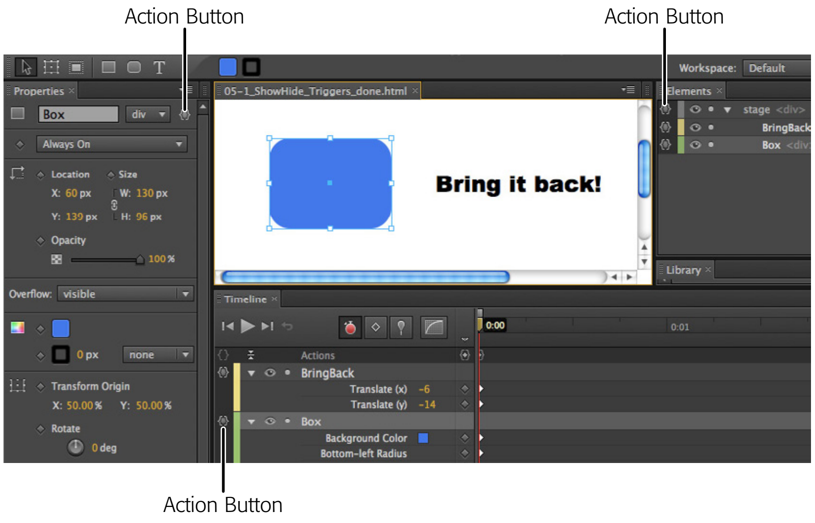


FIGURE 5-2

The first step in creating a trigger/action combo is to click an Open Actions button next to an element that's in your animation. Open Actions buttons look like small brackets.

NOTE When Edge adds triggers and actions to your animation, it's actually writing JavaScript/jQuery code. You view this code from the comfort of a panel with buttons. This arrangement shields you from some of the nitty-gritty details, but if you want to roll up your sleeves and make changes to the code, you can do that, too. Wondering why Edge uses curly brackets for the Open Actions icons? It's appropriate because JavaScript uses curly brackets to enclose the methods and functions that provide action.

Trigger Your First Action

The easiest way to understand how to use triggers and actions is to put them to work. Here's a bare-bones exercise that shows how to attach a trigger and an action to an element on the stage.

1. Go to File→New to create a new Edge project. With the stage selected, in the Properties panel's Title box, enter the name *Action Packed*.

   You don't have to name the stage element for your animation to work; however, the stage title is displayed as the page title in web browsers. If you don't specify a title, you see *Untitled* in the tab for the page.

2. Choose File→Save and save your project as *Action Packed* in an empty folder.

   Now you can save the file quickly with a simple Ctrl+S (⌘-S).

3. With the Rounded Rectangle Tool (R) create a rectangular object and give it the ID *Box*.

   Size, color, and other properties don't matter much for this quick exercise.

4. Right-click (or Control-click) on Box and choose Open Actions for "Box" as shown in *Figure 5-3*.

   The Actions panel for Box opens with the triggers menu displayed, as shown in *Figure 5-4*.

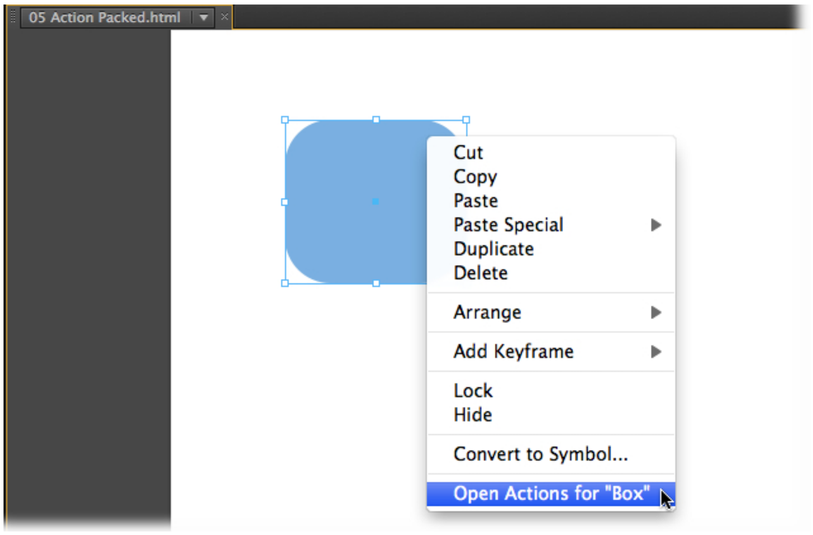


FIGURE 5-3

The quickest way to add a trigger to elements that are visible on the stage is to right-click the element and then choose the Open Actions option, as shown here.

5. From the Triggers menu, choose *click*.

   The *click* trigger is the first option on the triggers menu. If the triggers list closes before you select a trigger, click the + button to show the list again, as shown in *Figure 5-6*. After you choose *click*, the menu disappears, and some text is automatically added to the Box actions. It reads:

   ```
   // insert code for mouse clicks here
   ```

   This text is Edge's way of prompting you, explaining what you should do next. It's actually a comment. Comments in JavaScript code don't make anything happen; they're used to provide information. Edge displays comments in green. For more details, see the box on page 98.

TIP If you'd rather not see Edge's comments, you can turn them off by clicking the button in the upper right corner of the Actions panel and turning off "Include Snippet Comments."

6. On the right side of the Actions panel, click Hide Element.

   The JavaScript/jQuery code to hide an element is displayed in the Actions panel. As explained in the box on page 98, comments are displayed in green text. That means the only line of active code in the panel is the line that reads:

   ```
   sym.$("Text1").hide();
   ```

   The word "Text1" is automatically selected, because you need to change this word for your code to work. You need to tell Edge exactly which element you want to hide.

7. Replace "Text1" with *Box*.

   Make sure that the word Box remains inside the quotes, like so:

   ```
   sym.$("Box").hide();
   ```

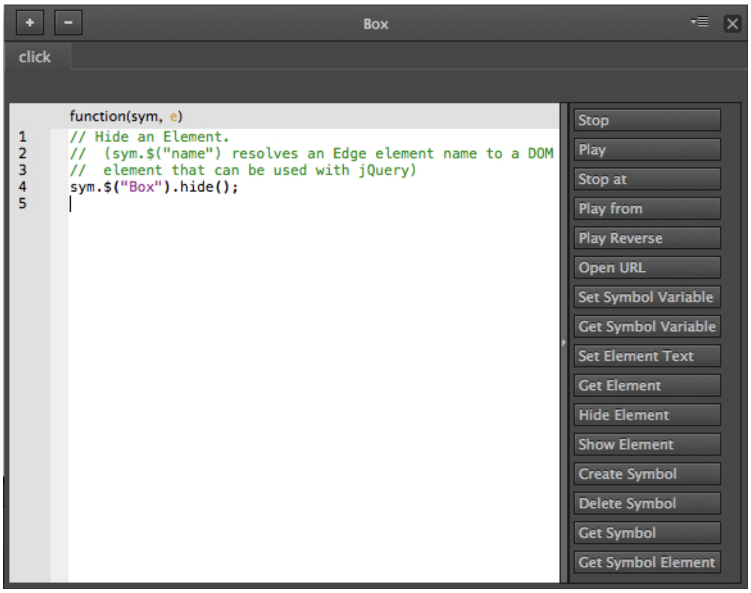
   When you're finished, the Actions window looks like *Figure 5-4*.

8. In the upper-right corner of the Actions panel, click the X button.

   The Actions panel for Box closes.

9. Press Ctrl+S (⌘-S) to save your work. Then press Ctrl+Enter (⌘-Return) to preview your project in a browser.

   To test your triggers and actions, you need to preview the project in a browser. Edge's workspace doesn't respond to clicks and other triggers.



```
function(sym, e)
// Hide an Element.
//  (sym.$("name") resolves an Edge element name to a DOM
//  element that can be used with jQuery)
sym.$("Box").hide();
```

FIGURE 5-4

When you've finished tweaking the code for the Box element, the Actions window looks like this. You use the buttons along the right side of the Actions window to add actions (JavaScript/jQuery code) to your project.

When you test this first trigger and action project, you see your web page called Action Packed. Your rectangle is displayed on the page, and when you click it, the rectangle disappears from view. If your version isn't working as advertised, go back and double-check that single line of code (step 7). Make sure the spelling and capitalization are correct. JavaScript and jQuery pay attention to capitalization.

Even when it works properly, the project at this stage is a little underwhelming, but it does hold promise for the future. In the next section, you'll add another element and more triggers. You'll see how a trigger in one element can affect another element.

NOTE At this point, don't worry too much about parsing the JavaScript/jQuery code. Chapter 8 provides more complete details.

UP TO SPEED

Many Ways to Comment

CSS and JavaScript use a common method to add comments to code. The system is also used by C and other programming languages. There are two types of comments: line comments and block comments. The comment shown in step 5 on page 88 is a line comment. When CSS or JavaScript code interpreters see two slashes (//) they ignore the remainder of that line. That means you can add descriptive text or anything else at the end of the line. For example:

```
sym.$("Box").hide(); //hides "Box"
```

Edge often places // at the beginning of a line, meaning that the whole line is ignored. Usually these comments prompt you on what to do next or explain the purpose of code inserted using the action buttons. When you come back to the code months later, you'll have a handy explanation about how it works. Edge uses green text for comments. This makes it easier for you to quickly differentiate between active code and comments.

When you want to add several lines of comments, you can use block comments. Block comments begin with /* and end with */. Everything in between these markers is considered a comment. JavaScript and CSS ignore any text that appears between these marks. Often, block comments are used at the beginning of a document to provide details about the project and perhaps information about the person or company that developed the code.

There's another popular use for comments among coders. Comments are frequently used to turn chunks of code on and off, while developers test and work on a project. Suppose you have a line of code that hides *Box* when you click on it. You're testing another part of your program and temporarily you don't want the box-hiding function to work. You don't want to delete the code, because you'll want to turn it back on when you're done testing. The solution is to place // in front of the line to turn it into a comment. In coder-speak, you're "commenting out" the line. When you're ready to bring it back, just delete the // marks. If you've got a big chunk of code you want to turn off, you can use the block quote marks.

Triggering Actions in Other Elements

The exercise on page 95 provided a bare-bones example for attaching a trigger to an element and then specifying an action that changed that element. In that case, the *click* trigger was attached to a rectangle named Box. Then the action code made the box *hide*, so Box was no longer visible on the stage. Fine as far as it goes, but that leaves the audience with an empty stage and nothing left to do. For a next step, you can add another element to the stage—a text box that appears when the box disappears. With a little trigger/action magic, the text can make the hidden Box element reappear. The obvious difference in this example is that the trigger in one element makes a change to a different element.

This exercise continues the Action Packed project started on page 95.

1. If necessary, go to File→Open to open the *Action Packed* project.

   The elements in this project are the stage (named Action Packed) and a rectangle (named Box). A *click* trigger and Hide action are already defined for Box.

2. With the text tool (T) create a text box and type *Bring it back!* Name the text box *BringBack*.

   Like the rectangle, the text box is an element. It's listed in the Elements panel and has a number of properties. You can create triggers and actions for a text box.

3. Right-click the text box named BringBack and choose Open Actions for "Bring-Back."

   The Actions panel for BringBack appears, and you're prompted to choose a trigger, as shown in *Figure 5-5*.

4. Choose the *click* trigger.

   The triggers menu disappears, and you see the actions listed on the right side of the Actions panel.

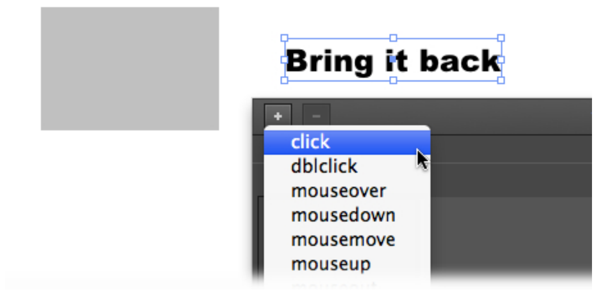


FIGURE 5-5

The "Bring it back" text shown here is about to get a click trigger of its own. The action attached to this trigger will change the visibility of the rectangle. Even though triggers are attached to an element, the actions can affect other elements and aspects of your animation.

5. Click the Show button.

   Comments and code appear in the Actions box. Again, the word "Text1" is preselected. This is Edge's subtle hint that you need to edit the code at this spot. You need to tell Edge what you want to show.

6. Replace Text1 with *Box*.

   You want the hidden Box to reappear when someone clicks "Bring it back!" Remember that "Box" needs to remain inside the quotes like this:

   ```
   sym.$("Box").show();
   ```

7. Go to File→Save. Then press Ctrl+Enter (⌘-Return) to preview your progress.

   If all goes according to plan when you test Action Packed in your browser, Box disappears when you click it. Then when you click on the *Bring it back!* text, the Box graphic reappears.

Now, at least, there's a way to bring Box back after it disappears. The example also proves that a trigger in one element (the text box *BringBack*) can make changes to a different element (the rectangle *Box*). The technique followed the same development process: Choose an element, set the trigger, and specify the action. Still, there's something a little odd about this animation, as simple as it is. It's strange to have the text saying, "Bring it Back!" while the box is visible. It would be better if the text was hidden when the box was visible and vice versa. That's the task for the next section, where you'll also learn how to use the Timeline to trigger an action.

Triggers and Actions for the Stage and Timeline

The two earlier examples showed how to attach triggers and actions to elements on the stage. Even though they're different types of elements, the rectangle and the text box used very similar triggers and actions. Not so with the two objects next up for discussion: the *stage* and the *Timeline*. The stage shows up in the Elements panel, but because it is a container for the other elements, it has different properties from a graphic or text box. The same is true when it comes to triggers for the stage. You open the stage's actions by clicking the bracket-shaped Open Actions button next to the stage in Elements, Properties, or the Timeline. You'll see a list of triggers, including some that weren't displayed for the rectangle and text box.

- **compositionReady:** Triggers when the various parts of the Edge composition are read by the browser.
- **scroll:** Triggers when the page scrolls.
- **keydown:** Triggers when a key on the keyboard is pressed down.
- **keyup:** Triggers when a pressed key is released.

The Timeline also has its own unique triggers. Technically, the Timeline isn't even an element. It's not listed in the Elements panel. On the other hand, the Timeline does generate events, and those events can be used to trigger actions. To review the triggers for the Timeline, click the bracket-shaped Open Actions button next to the word "Actions" in the Timeline. You'll find these triggers:

- **update:** Triggers when the stage changes.
- **play:** Triggers when the Timeline begins to play.
- **complete:** Triggers when the Timeline reaches the end.
- **stop:** Triggers when the Timeline stops playing, even if it's not the end of the Timeline.

In the case of the Action Packed project, you want Box to be visible at the start, but not the "Bring it Back!" text. Reviewing the stage and Timeline actions, there seem to be two likely candidates for hiding something at the start. The stage's compositionReady trigger runs at the start of an animation. Likewise, the Timeline's play trigger runs when an animation begins to play.

Spoiler alert! compositionReady is the better choice; however, you can see why if you try the Timeline's play trigger first. Not only that, but you'll also learn how to remove an unwanted trigger from your project. Follow these steps to test the Timeline's play trigger. With the Action Packed project open in Edge, follow these steps:

1. In the Timeline, next to Actions, click the Open Actions button.

   The Default Timeline Actions panel opens, and the menu shows four triggers: "update," "play," "complete," and "stop."

2. Choose the play trigger from the list.

   If the list disappears before you make your choice, click the + button in the upper-left corner.

3. From the list of actions on the right, click Hide Element.

   The generic code to hide an element is displayed in the Actions panel. The word "Text1" is preselected, because you need to change this text to indicate the element that is to be hidden.

4. Replace "Text1" with *BringBack*.

   BringBack is the ID or name for the text box that says, "Bring it Back!"

5. Click the X button in the upper-right corner of the Actions panel.

   The Default Timeline Actions panel closes.

6. Press Ctrl+S (⌘-S) to save your project. Then press Ctrl+Enter (⌘-Return) to preview it.

When you preview your animation, you may see a little flicker at the beginning where the BringBack text appears briefly on the stage before it's hidden. This is one of the reasons why it's not the best option for this job. The other reason is that play trigger goes into action whenever the Timeline begins to play—not just at the beginning of the Timeline. The compositionReady trigger works differently. It triggers when the Edge project is first loaded into a browser. That means it triggers once at the start. As an added benefit for this job, the text is less likely to be displayed and then hidden.

Deleting Triggers and Actions

Removing a trigger or action from your project is as easy as clicking a button. In the case of the Action Packed project, you want to remove the *play* trigger and its related action from the Timeline. Here are the steps:

1. **In the Timeline next to Actions, click the Open Actions button.**

   The Default Timeline Actions panel opens. The title bar of the Actions panel names the element. In this case, it says Default Timeline. Tabs near the top of the panel show triggers that are attached to the element. In this case, there's only one trigger in use: "play."

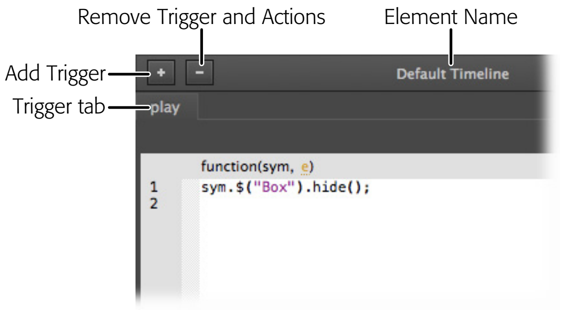
2. **With the Play tab selected, click the minus button (–) in the upper-left corner.**

   Unless you have more than one trigger attached to the Timeline, the Play tab is already selected. When you click the minus button (–), the entire tab is removed and the Actions panel is empty.

3. **Click the X button in the upper-right corner.**

   The Actions panel closes.

Edge doesn't give you a warning or an alert when you delete a trigger and its accompanying actions. Just whoosh and everything's gone. However, if you make a mistake, you can bring the trigger and actions back as long as you do it right away. Just use your good old friend Edit→Undo (Ctrl+Z in Windows or ⌘+Z on a Mac).



```
function(sym, e)
sym.$("Box").hide();
```

FIGURE 5-6

Use the plus button (+) to add triggers to an element. Use the minus button (–) to remove them. Each trigger that's attached to an element is shown in a separate tab. Click the tab to see the actions code that runs when the trigger is pulled.

Adding Triggers to the Stage

Having tried and removed a Timeline trigger to hide the BringBack text box, it's time to try the Stage's *compositionReady* trigger. *compositionReady* runs once when the HTML code is read by a web browser. It sounds like the perfect tool to hide an element at the start. Then you can use another trigger to show the element at the right moment. By now you probably know the drill for adding a trigger and the Hide action; however, here are the steps as a reminder:

1. In the Elements panel, click the Open Actions button next to the stage.

   The many triggers for stage appear in the open menu. Fortunately, *compositionReady* is at the top of the list.

2. Click *compositionReady.*

   The *compositionReady* trigger tab is created in the Actions panel, and you're ready to code.

3. On the right, click the Hide Element button.

   The code for the generic Hide Element action appears in the panel.

4. Replace "Text1" with "*BringBack*".

   BringBack is the ID for the text box. Remember to leave the quotes around "BringBack."

5. In the upper-right corner, click the X button.

   The Actions panel closes.

6. Press Ctrl+S (⌘-S) and then press Ctrl+Enter (⌘-Return).

   The first command saves your project. The second previews the project in your browser.

Now the text box is hidden at the beginning of your animation. However, when you click Box, the box disappears and there's no way to bring it back. You need to tell Edge exactly when you want the text box to show up.

Editing an Action

It just so happens, you've already created the proper trigger to make the BringBack text appear. It's Box's *click* trigger, the same trigger that makes Box disappear. All you need to do is add code to show the BringBack text box. You can tell that Box already has at least one trigger and action, because the bracket-shaped button next it in Elements and the Timeline is bright white and not grayed out. Here are the steps to open the actions and make some changes:

1. In Elements, click the Open Actions button next to Box.

   The Box Actions panel opens. In this case, you're not prompted to choose a trigger because Box already has a trigger: *click*.

2. In the *click* actions code, click to put the editing cursor on an empty line at the bottom.

   The actions code pane is a workplace. You can add more actions using the buttons on the right, or if you're comfortable speaking jQuery, you can add your own code.

3. On the right, click the Show Element button.

   As usual, the generic Show code appears, and you need to specify what element is to be shown.

4. Replace "Text1" with *BringBack*.

   "Text1" is placeholder text. BringBack is the ID of the text box that says, "Bring it Back!"

5. In the upper-right corner, click the X button.

   The Actions panel closes.

6. Press Ctrl+S (⌘-S) and then press Ctrl+Enter (⌘-Return).

   The first command saves your project. The second previews the project in your browser.

Now, when you preview the animation, it's behaving better. When it begins, the BringBack text is hidden. A click on Box performs two actions. It hides Box and displays BringBack. It's clear to the viewer what must be done. Click on the text, and the box comes back. Just one issue: The text should hide when Box shows. Time for another quick edit.

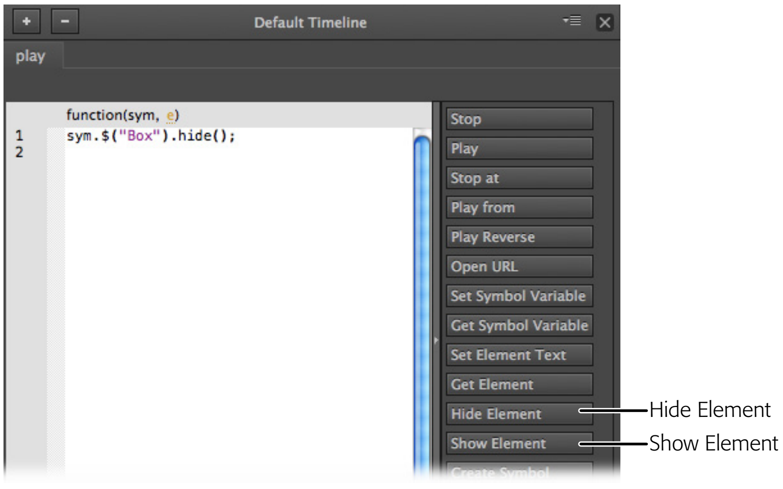
1. Right-click (Control-click) the text box BringBack and choose Open Actions for "BringBack."
2. With the Click tab selected, place the editing cursor at the bottom of the code.
3. Click the Hide Element actions button.
4. Replace "Text1" with *"BringBack"*.
5. Save and preview your project.

It may not be as complex as Google Docs or Netflix's movie queue, but this time, your project works well. Elements are showing up and disappearing as they should. It's a pretty simple trick, but you'd be surprised how many interesting web pages are based on making elements show and hide. What's more, as you'll learn on page 110, it's easy to substitute those show and hide methods with something a little more interesting, like fade in and fade out or other effects.

NOTE Want to compare your project to a completed version of this exercise? Download *05-1_ShowHide_Triggers_done* from the Missing CD at *www.missingmanuals.com/cds/edgepv5mm*.

Just to recap, here are some of the trigger/action concepts discussed in the exercise that began back on page 95:

- There are three main steps to interactivity: Choose an element. Attach a trigger. Specify an action.
- A trigger is an event, like a mouse click, that's attached to an element.
- Actions are defined using JavaScript/jQuery code.
- Actions can affect elements other than the one with the trigger.
- A single trigger can lead to more than one action.
- The stage and the Timeline have triggers unique to their role in an HTML document.
- You can add, remove, and edit triggers from the Actions panel for each element (*Figure 5-7*).
- You need to preview your project in a browser to test triggers and actions.



```
function(sym, e)
sym.$("Box").hide();
```

FIGURE 5-7

In Edge, Actions panels share the same features whether they're attached to elements on the stage or the Timeline. Use the + and – buttons to add and remove triggers. Use the buttons on the right to add prebuilt code for specific actions. Your editable code appears in the main part of the box.

Timeline Triggers and Tricks

When you work in Edge, you've always got that Timeline down there at the bottom of the screen. It's keeping track of the elements, their positions on the stage, and any visual changes that you want. In some projects, the Timeline is less relevant. For example, consider the previous exercise, where the playhead never leaves the 0:00 mark. In that case, changes are entirely controlled by triggers and actions. It's also worth noting that that means change and timing are, for the most part, controlled by the audience. Earlier projects like the slideshow (page 43) used the Timeline to control changes and timing. There were no triggers and actions. Naturally, you're not forced to choose one method or the other. You can create masterpieces when you combine the strengths of Timeline animation with triggers and actions.

Adding a Trigger to Loop Your Animation

As gatekeeper to your animation's timing, the Timeline has special triggers like "play," which fires when the animation starts playing. One of the most used Timeline triggers is "complete," which fires when the playhead reaches the end of the Timeline. So if you want to make an animation loop back to the beginning, you use the complete trigger and add code to send the playhead back to 0:00. Want to give it a try? Roll up your sleeves and grab the files and folders in *05-2_Timeline_Trigger.*

Open the web page, *02_Timeline_Trigger.html*, in a browser, and you see that it's a simple counter. Every half-second, a new number appears on the stage. If you dig into the Edge project, you see that the appearance and disappearance of the numbers are controlled by keyframes that toggle the opacity properties of each number. In other words, it's a simple animation, and at this point there's no trigger/action magic involved. That's about to change. Here are the steps to add a Timeline trigger that makes the animation loop continuously:

1. Open *05-2_Timeline_Trigger.edge* in Edge.

   There are five elements placed on the stage. They are named num_1 through num_5.

NOTE Element names can't begin with a number, so you need to place at least one letter at the beginning.

2. On the Timeline, next to the word "Actions," click the Open Timeline Actions button and choose the complete trigger.

   The complete tab appears in the Actions panel, and a comment prompts you to "Insert code to be run at the end of the timeline."

3. On the right, from the list of actions, choose "Play from."

   Edge inserts a comment and a line of code:

   ```
   // play the timeline from the given position (ms or label)
   sym.play(1000);
   ```

   The code isn't too hard to parse here: "sym" refers to the Timeline itself; "play" is a method that tells the Timeline to run or play. The number inside of the parentheses tells the Timeline where it should start playing. Edge pops the "1000" in there simply as a placeholder. The unit of time measurement is one one-thousandth of a second. So 1000 is equivalent to the 1-second mark. If you preview your animation at this point, after playing once, the animation jumps to the 1-second mark and displays the number 3. Then it continues looping by always jumping back to the 1-second mark.

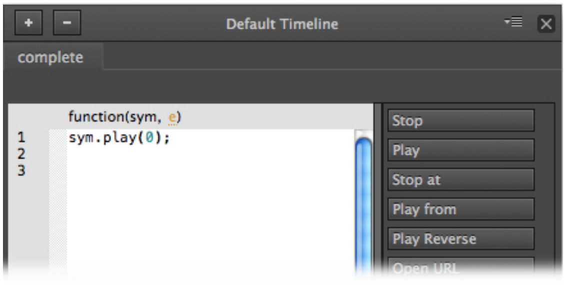
4. Change the 1000 inside the parentheses to *0*, as shown in *Figure 5-8*.

   This command sends the playhead back to the 0:00 mark on the Timeline:

   ```
   sym.play(0);
   ```

5. Press Ctrl+S (⌘-S) to save and Ctrl+Enter (⌘-Return) to play.

   Your animation loops continuously.



```
function(sym, e)
sym.play(0);
```

FIGURE 5-8

The code shown here in the Actions panel was inserted using the "Play from" button. Originally, the placeholder text "1000" was in the parentheses. By changing that to 0, the code sends the playhead to the beginning of the Timeline.

By using the complete trigger on the Timeline and adding the simple line of code shown in step 4, you can make any of your Edge animations loop. As you saw at the start, you can send the playhead to any point on the Timeline using the thousandth-of-a-second unit of measurement. Want the playhead to go to the 6-second mark? Just pop *6000* in between those parentheses.

Using Labels in Your Timeline Code

You're not limited to specifying Timeline locations by thousandths of a second. A far easier way to identify a point on the Timeline is to use a label, as described back on page 77. A label is a word that appears on the Timeline ruler, and for most human beings, labels are more descriptive and easier to remember than numbers representing fractions of a second. Suppose you want to add labels to the counters' Timeline for each spot where the number changes. Start by moving the playhead to 0:00, and then press Ctrl+L (⌘-L) to insert a label. Type a name, perhaps the word *one*. For the next number, move the playhead half a second down the Timeline where the number on the stage changes and repeat the process. Once you're done with all five labels, the Timeline looks like *Figure 5-9*.

Now, using the same Timeline trigger ("complete") and action ("Play from"), you can send the playhead back to any of the labels. For example, to send the playhead to the label "two," edit the action so it looks like this:

```
sym.play("two");
```

Notice that the label is inside the quotation marks. When you reference a label, it must always be inside quotes. When you're developing larger and more complex Edge projects, you'll find plenty of opportunities to use Timeline labels as destinations for actions.

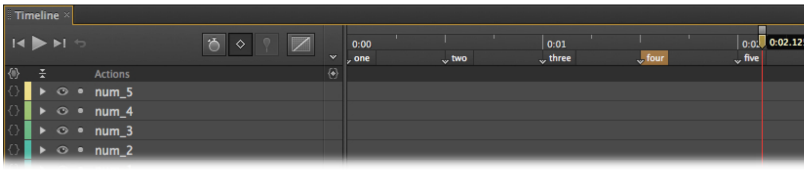


FIGURE 5-9

Labels on the Timeline mark each point where the counter changes from one number to the next. You can use labels in your actions to move the playhead to a specific position.

Adding Triggers to a Point in Time

You can add triggers to a specific point along the Timeline and then specify an action for that point in time. Suppose you want to briefly display a caption in the middle of your animation. You can place a trigger in the Timeline that "shows" the caption, and then a moment later, you place another trigger that hides the text box with the caption. If you want to try this experiment, you can use the Timeline Trigger exercise beginning on page 106. Here are the steps to display a caption for half a second in the middle of the counter. Suppose you're overly fond of the number 4, and you want to display the message "Great Number" while the 4 is displayed. Here are the steps:

1. With your Timeline Trigger project open, use the text tool (T) to create a text box that says, "Great Number." Name the text box *Great*.

   Format the text so that it looks good with your counter numbers. For text formatting details, see page 51.

2. Make sure that Auto-Keyframe Properties and Create Smooth Transitions are toggled off. Drag the playhead to 0:00.

   At this point in the animation, you want the text box hidden.

3. Go to Timeline→Insert Trigger.

   An Actions panel titled "Trigger" opens. You see a panel like this each time you add a Timeline trigger, but each trigger has its own panel.

4. Click the Hide Element action button. Then, in the Hide Element code, replace "Text1" with *Great*. Close the actions box.

   As shown in the earlier exercises, this code hides the text box named Great. The action should read:

```
sym.$("Great").hide();
```

When you close the actions, check the Timeline under the "one" label, and you see a trigger icon (*Figure 5-10*). This particular trigger is only half visible because it's positioned at the start of the Timeline.

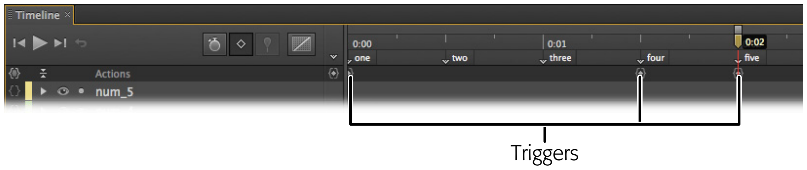


FIGURE 5-10

Timeline triggers look similar to property key-frames in the Timeline. The difference is that they all appear on the Actions row, and they have brackets surrounding the diamond marker.

5. Drag the playhead to the "four" label and press Ctrl+T (⌘-T).

   Another trigger actions box opens. Ctrl+T (⌘-T) is the shortcut key to create Timeline actions.

6. Click the Show button, and after you see the new code, replace "Text1" with *Great*. Close the actions box.

   This trigger shows the caption "Great Number." Another action keyframe appears in the Timeline.

7. Move the playhead to 0:02, the point with the "five" label. Press Ctrl+T (⌘-T) to create a Timeline trigger. Add hide code just like the code in step 4.

   At the three points in the Timeline triggers, you see triggers ready to perform their accompanying actions.

8. Press Ctrl+S (⌘-S) to save and Ctrl+Enter (⌘-Return) to play.

   Your animation plays. While the number 4 is displayed, the caption also shows with the words "Great Number."

Keep in mind that you need to preview and test your work in a browser when you're using triggers and actions. Even something as simple as Show and Hide actions require a browser to read and interpret the JavaScript/jQuery code. The finished project can be found with the Missing CD files. It is named *05-3_Timeline_Trigger_done.*

TIP Want more practice with Timeline triggers and actions? Try rebuilding the counter so triggers control the numbers' visibility. First remove all the opacity keyframes. Then add triggers and actions to the Timeline to take over the Show and Hide duties.

Sliding Show Revisited

The exercise back on page 43 showed how to create a slideshow that automatically swapped images using transitions that included changes in locations and transparency. Each slide faded away and actually slid off the stage. Using the same images, you can create a different type of slideshow. Suppose you want the viewer to be in control. Each time she clicks the mouse, the slide changes instantly, moving to the next slide. If she's on the last image in the show, a mouse click takes her back to the beginning. You can find the files for this project on the Missing CD at *www.missingmanuals.com/cds/edgepv5mm*. Find *05-4_Click_Show* for the folders and files for this exercise. If you want to see the finished project, check out *05-5_Click_Show _done*.

Follow these steps to create a clickable slideshow:

1. In Edge, go to File→Open and then select *05-4_Click_Show.edge.*

   The Elements panel for this project shows a stage and three images: squirrel, house, and bike. All three images are already positioned on the the stage. Each image is 600 x 400 px, the same size as the stage.

TIP You can examine each of the images by toggling the show/hide buttons for the elements, as explained on page 43.

2. Press I to turn on Instant Transitions.

   The button in the Timeline appears pushed in when this feature is turned on.

3. In Elements, select the squirrel, house, and bike elements. Then in Properties, click the Add Keyframe button next to Location.

   This adds keyframes to the beginning of the Timeline to lock in the initial position for all the photos.

4. With the playhead at 0:00, press Ctrl+L (⌘-L) to create a label named *squirrel*. At 0:01, create a label named *house*. At 0:02, create a label named *bike*.

   When you're done, the Timeline should look like *Figure 5-11*.

5. Move the playhead to 0:00. Then click the Insert Trigger button shown in *Figure 5-11*, and choose Stop from the available actions.

   When you click the Insert Trigger button, a panel opens where you can add and edit the JavaScript/jQuery code that controls your animation. The button on the Actions row opens the code window for the Timeline trigger, shown in *Figure 5-12*.

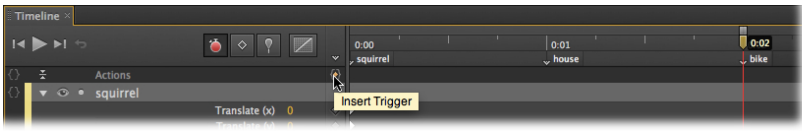
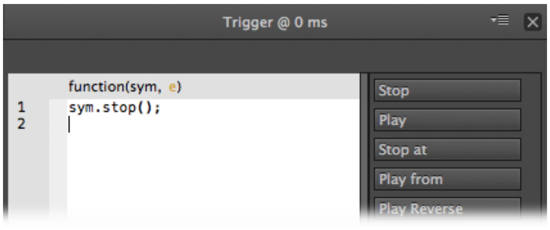


FIGURE 5-11

In this case, it doesn't really matter how far apart you place the labels on the Timeline, because there will be no transition. JavaScript/jQuery code will make the playhead jump from label to label. Just give yourself enough space to separate the labels for easy reading.



```
function(sym, e)
sym.stop();
```

FIGURE 5-12

The code in this Trigger window controls the main Timeline at a specific point in time. In effect, it stops the animation from running at the very start. That way the first image stays on the stage until the viewer clicks on it.

6. Right-click (Control-click) the squirrel photo on the stage, and then choose Open Actions for "squirrel."

   The actions panel labeled "squirrel" opens, where you can write or insert JavaScript/jQuery code. Initially, several trigger options, like *click*, *dblclick*, and *mouseover*, are displayed. If you move the box, the list may disappear. You can always bring it back by clicking the + button in the upper-left corner.

7. Choose "click" from the triggers list.

   The panel changes, displaying the actions that can be associated with the click trigger, as shown in *Figure 5-13*.

8. Choose "Stop at."

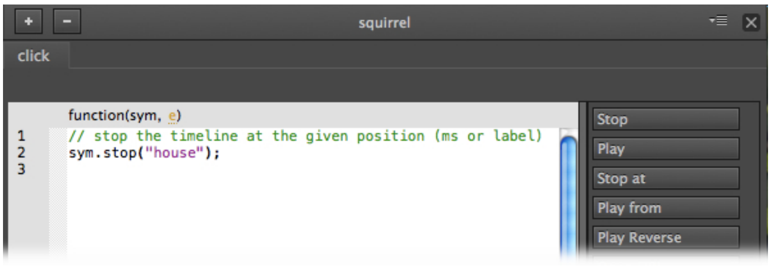
   The code for the "Stop at" method is added to the squirrel actions panel. It reads:

   ```
   // stop the timeline at the given position (ms or label)
   sym.stop(1000);
   ```

The first line is a comment describing the "Stop at" method. The second is the code that sends the playhead to a particular point in the Timeline and stops playback. The 1000 inside the parentheses is a numerical reference to a point on the Timeline. In the next step, you'll replace that reference with a reference to one of the labels you created earlier.

9. Replace "1000" in the code with *house*. When you're done, click the X button in the upper-right corner of the box to close it.

   Don't forget to include the quote marks (") as shown in *Figure 5-13*. When you reference a label, you must put the label in quotes.



```
function(sym, e)
// stop the timeline at the given position (ms or label)
sym.stop("house");
```

FIGURE 5-13

You can associate prebuilt actions with a trigger. It's simply a matter of clicking a button. The code is then automatically added to the Actions panel. In this exercise, you use the "Go To and Stop" action to control your slideshow.

10. In Elements, click the show/hide button (eyeball) next to squirrel.

    The squirrel picture is hidden from the stage, and the house picture is visible.

11. Repeat steps 5–8 for the house image, creating a *click* trigger and a "Stop at" action. In step 8, instead of replacing "1000" with the *house* label, use the *bike* label.

12. Hide the house photo. Then repeat steps 5–8 for the bike photo. In the "Stop at" method, use the *squirrel* label.

    The method now sends the playhead back to the first label on the Timeline: *squirrel*. Next, you'll set up the Timeline to display the proper image at each label.

13. In Elements, click the show/hide buttons to make the squirrel and farmhouse visible.

    A diamond-shaped property keyframe appears on the squirrel layer. If the Generate Smooth Transitions button is turned off, as recommended in step 2, you don't see any transition bars. Keep in mind that the show/hide buttons affect the elements only during design time. They have no effect on what your audience sees when they view your project in a browser. You'll fix that in the next few steps.

14. Move the playhead to the *house* label. Then drag the squirrel photo off the stage completely.

    It doesn't matter where you drag the photo. The audience won't see its movement because there's no transition. The playhead just jumps from label to label. At this point in the animation, the audience will see the house photo.

15. Move the playhead to the *bike* label. Then drag the house photo off the stage completely.

    At this point in the animation, the audience will see the bike photo.

16. In Elements, select the stage. Then, in Properties, make sure the Overflow property is set to hidden.

    If Overflow is set to "visible," your audience will see the photos that are offstage.

17. Choose File→Preview In Browser to see your slideshow in action.

    You need to view the slideshow in a browser to check the clicking action. If you press Play to view the slideshow within Edge, you won't get the benefit of the JavaScript code that controls the playhead.

At this point, you're probably thinking of different ways you can create a slideshow for your own photo library. You can tackle the project in a number of ways, and the example above is just one solution. For example, using Timeline labels, triggers, and actions, you can create a button interface for your slideshow. Buttons could advance the show forward and backward. Other buttons could jump to the beginning, end, or special sections of your slideshow. You could combine Timeline animation effects with Timeline trigger controls, creating a slideshow that has animated effects and gives the audience control over the experience.

Non-Linear Thinking and Design

Up to now, most of the projects in this book have been fairly linear, moving along the Timeline in a natural left-to-right fashion. When you start using labels, Timeline triggers, and the other tools covered in this chapter, you aren't limited to a simple linear Timeline. Suppose you're developing a graphic novel for the Web. The novel has three alternate endings, and at some point you let your audience choose one of the endings. See *Figure 5-14*. All you need to do is stop the Timeline at a certain point. Create three buttons or some other type of widget representing the three different endings. Then connect triggers and actions to the buttons that lead to different labeled points on the Timeline. Want to get more elaborate? You could develop an entire adventure-style game, where the player's decisions lead to different challenges. In that case, the action could branch out in all different directions. Certain user actions could make the game loop back to a previous point.

For these kinds of projects, you need to think of the Timeline as a kind of random access storage device, like a computer hard drive. Using triggers and actions, you access the parts of the storage that you need at a given moment. One portion of the Timeline might hold a single text box, like the answer to a quiz question. Or it might contain an entire animated sequence for a graphic novel. After you've used (or read) that portion of the Timeline, use Timeline triggers to send the playhead back to another point in the Timeline.

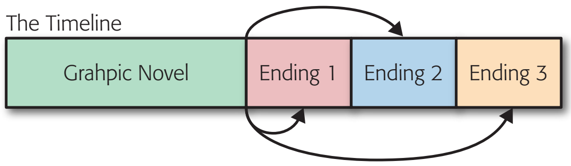


FIGURE 5-14
Just because it's called a Timeline, that doesn't mean you have to play an animation in a linear fashion. Here's a diagram of a graphic novel with alternate endings.

Triggers for iPhones and Androids

So far, the activities in this chapter have focused on just a few common triggers and actions. When you get into developing your own projects, you're likely to do the same. The simple click is probably the most used trigger on the planet, because it's the simplest form of interaction. Showing and hiding elements and jumping to spots on the Timeline are all natural actions for a development system like Edge. However, time marches on, and we all browse the Web using iPhones, Androids, and tablets of all kinds. One of the great benefits of HTML5 is that it's been designed with the handheld revolution in mind. That's true of Edge, too.

When you open the triggers menu for an element like a rectangle, you might notice that they are divided into two groups (*Figure 5-15*). At the top, you have standard desktop, mouse-like triggers, like *click* and *dblclick*. As a developer, you'll be glad to hear that these commands work on touchscreens as well as mouse-operated computers. As you might guess, the mouse commands like mouseover and mouseout are exclusive to mice, because there aren't any related touchscreen triggers. At the bottom, you have three triggers that are used with touchscreens: *touchstart*, *touchmove,* and *touchend*.

If you want to reach the widest audience, you may as well go ahead and start thinking about developing apps that work equally well for mobile devices and traditional computers. These days you never know whether someone will be viewing your work on a 30-inch widescreen monitor or a 4.5-inch iPhone.

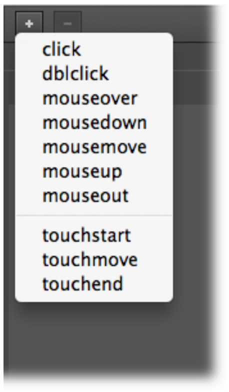


FIGURE 5-15

The triggers at the top of the list work for computers, but some—like click and double-click—work for phones and tablets, too. The bottom group of triggers are touchscreen specific.

PART

3

Edge with HTML5 and JavaScript

CHAPTER

6

Working Smart with Symbols

There's one trait that animators, web builders, and programmers all share. They hate to do the same job more than once. In each of these crafts, there are tools and techniques that help you minimize the grunt work and maximize the time available for creativity. In Edge, *symbols* work that way. You build an element once, and then you can use it many times. Symbols make your web page more efficient, too. If you're building a scene with a couple hundred raindrops, all you need is one definition of a raindrop in order to fill the sky with them. You can even change the size, rotation, and opacity of the individual instances to add variety to your scene.

This chapter gives you all the details about creating, using, and editing symbols. You'll also learn some tricks for working with Edge's Library panel—the storage barn for symbols and other project assets.

About Symbols

Copying and pasting is the most obvious way to reuse something you've created, but while that time-honored technique saves time, it doesn't save *space*. Say, for example, you need to show a swarm of cockroaches in the Edge advertisement you're creating for New and Improved Roach-B-Gone. You draw a single cockroach and then copy and paste it 100 times. Congratulations: You've got yourself 101 cockroaches... and one big, slow page download.

Instead, you should take that first cockroach and save it in Edge as a *symbol*. Symbols give you a way to reuse your work and keep your animation's finished file size down to a bare minimum. When you create a symbol, Edge stores the information for the symbol, or master copy, in your document as usual. But every time you create a copy (an *instance*) of that symbol, all Edge adds to your file is the information it needs to keep track of where you positioned that particular instance and any modifications you make to it on the stage.

So, to create the illusion of a swarm of roaches, you drag instances of the symbol onto the stage. Neither you nor Edge has to duplicate the work of drawing each roach. You can even vary the roach instances a little for variety and realism (so important in a pesticide ad) by changing their opacity, position, size, and even their skew. If symbols offered only file optimization, they'd be well worth using. But symbols give you additional benefits:

- **Grouping.** Keeping certain elements together, so you can operate on them all at once, helps you save time and stay organized. For example, in the case of those roaches, you can keep the body, legs, and antennae together in one discrete unit, making it much easier to move each insect around the stage.
- **Consistency.** By definition, all the instances of a symbol look pretty much the same. You can change certain instance characteristics—opacity and position on the stage, for example—but you can't redraw them without changing all instances of the symbol. Edge simply won't let you. (You can't turn one roach into a ladybug, for example.) For situations where you really need basic consistency among objects, symbols help save you from yourself.
- **Instantaneous update.** Suppose you want to change the roach color from brown to black. Edit the "master" roach stored in your Library, and Edge automatically updates all the instances of that symbol. So, for example, say you create a symbol showing the packaging for Roach-B-Gone. You use dozens and dozens of instances of the symbol throughout your animation, and then your boss tells you the marketing team has redesigned the packaging. If you'd used Copy and Paste to create all those boxes of Roach-B-Gone, you'd have to find and change each one manually. But with symbols, all you need to do is change the symbol in the Library. Edge automatically takes care of updating all the symbol's instances for you.

  Does that mean you can't make changes to *one* of the roaches already on the stage? No, not at all. You can change an instance without affecting any other instances or the symbol itself. (You can turn one roach light brown using opacity, for example, without affecting any of the dark brown roaches.)
- **Nesting.** Symbols can contain other symbols. Sticking symbols inside other symbols is called nesting, and it's a great way to create unique, complex-looking images for a fraction of the file size you'd need to create them individually. Suppose you've drawn the perfect bug eye. You can turn it into a symbol and place it inside your symbols for roaches, ladybugs, and any other insect you want.

Building with Symbols

Here's a mad scientist experiment. Build a roach using as few parts (symbols) as possible, using Edge to create all the elements. That means you're limited to Edge's primitive drawing tools: the rectangle and its cousin, the rounded rectangle. You're going to have to be a little creative. Need some guidance? Here are the steps for creating three symbols, using only Edge's drawing tools. Then you assemble those parts into a fairly respectable roach (if there actually is such a thing).

1. Create and save a new Edge project called Build-A-Roach.

   Make sure you save it in its own folder. After all, you don't want roaches all over the place.

2. Use the rounded rectangle tool to draw a roach body that's 200px long and 100px wide.

   Pick an ugly brown color for your bug—something like R=70, G=60, B=30 will do the trick.

TIP Even if your ultimate plan is to have lots of little roaches in your project, it's easier to build one large master roach. Working bigger gives you more mouse control over control handles and other objects you need to manipulate. You can always scale the roaches down when you add them to your project.

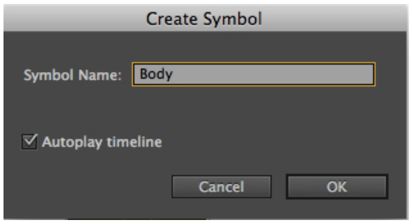
3. Use the border radii tools in Properties or the Transform Tool to make your rounded rectangle oval shaped and somewhat pointed on the ends—like a roach body.

   Roaches vary, so your bug body doesn't have to match this book's roaches.

TIP Sometimes when you're trying to draw, it works best if you turn off the smart guides. You can avoid some unwanted snapping as you position elements and bounding box control handles.

4. Right-click (Control-click) the body and choose Convert to Symbol from the shortcut menu.

   The Create Symbol dialog box, shown in *Figure 6-1*, appears. It has a Symbol Name text box and a Autoplay timeline check box.



Body

☑ Autoplay timeline

FIGURE 6-1

Every time you create a new symbol you need to give it a name. That name appears in the Library panel under Symbols. When you create instances of a symbol, you can give each instance a different Element ID. That way you can control instances independently.

5. Into the Symbol Name text box, type *Body*. Leave "Autoplay timeline" turned on. Press OK.

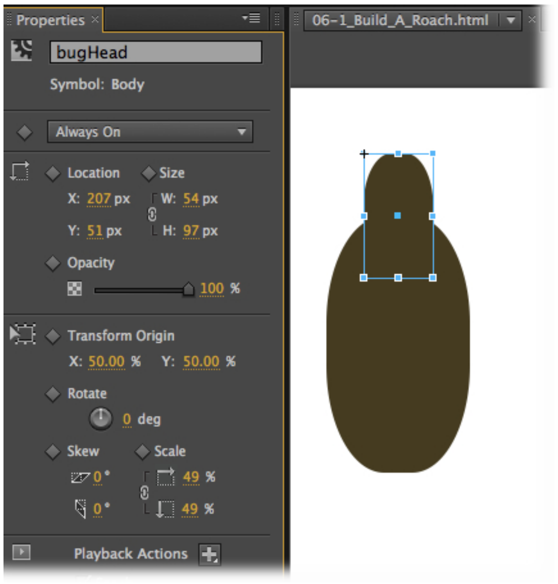
   You've transformed the simple rounded rectangle into a symbol, with all the rights and privileges that accrue to that new lofty status. How do you know it's a symbol and not just a run-of-the-mill rectangle? When it's selected, take a look at the icon next to its name in the Properties panel. The gear-shaped icon indicates Body is a symbol.

6. With the Body on the stage selected, in the Properties panel type bugBody in the ID box.

   You're going to use more than one instance of the Body symbol as you build your bug, so it's best to give each instance an appropriate ID.

7. Drag one more instance of Body to the stage and give it an ID of: bugHead. Then resize and reshape the instance into a bug head.

   You can drag the handles around the bugHead bounding box to reshape it or you can use any of the controls in the Properties panel shown in *Figure 6-2*.



bugHead
Symbol: Body
Always On
Location X: 207 px Y: 51 px
Size W: 54 px H: 97 px
Opacity 100 %
Transform Origin X: 50.00 % Y: 50.00 %
Rotate 0 deg
Skew 0° 0°
Scale 49 % 49 %

FIGURE 6-2

Note that the available properties are different for a symbol than they are for a rounded rectangle. For example, you can control Opacity, but you can't change the basic color. As shown at the bottom of the Properties panel, you can control playback actions because each symbol has its own timeline.

Building a Multipart Leg Symbol

At this point your roach consists of two parts made from one symbol. If you want to, you can continue to enhance your roach using the Body symbol. For example, you can make roach wings from the body symbol and ID them as leftWing and rightWing (no political puns intended). Then, use the Opacity, Size, and Skew properties to make the rounded rectangles look more wing-like. Wings aside, what your roach really needs is legs—six of them, so it can scurry across the floor when the light comes on. The trick is to create a single leg symbol that you can use in all the positions: left, right, front, back, and middle. Here's one solution:

1. Draw a long skinny leg-like rectangle. Set the background color to black.

   At this point, the limit of Edge's drawing tools becomes apparent—there's no line tool. You could turn to your favorite drawing program and import some artwork. But to meet the current Edge-only challenge, you can use skinny rectangles for your lines.

2. Add two more skinny rectangles to create a crooked roach leg.

   As you build the perfect roach leg, consider the different positions it'll appear in. A somewhat symmetrical leg may be best for all front/back and left/right purposes.

3. Select each of the three leg parts, then go Modify→Create Symbol.

   Modify→Create Symbol is simply another way to open the Create Symbol dialog box and turn a selection into a symbol. You could just as easily use the right-click (Control-click) method.

4. In the Symbol Name box, type *Leg*.

   You now have a Leg symbol and one instance of that symbol is on the stage.

5. Move the leg into position on your roach and change its ID to match its position.

   For example, the ID might be legLeftFront or legRightMiddle.

6. Add the rest of the legs to your roach.

   With a little creativity, you can provide six decent bug legs. If you want to give the front legs a look that's different from the others, try spinning them around 180 degrees.

Creating a Curved Line with a Rounded Rectangle

Your roach is now looking mobile. It still needs some antennae to feel its way around. If you want to really be economical, you can use the Leg symbol to make antennae. For exmaple, you can hide most of the leg under the body so that only one or two of its rectangles show. You could use size and skew properties so the antenna doesn't look leg-like. In reality though, most roaches have graceful, long curved antenna. So, for the sake of entomological realism, here are the steps for creating a curved line when your only drawing tool is a rounded rectangle.

1. Draw a long narrow rounded rectangle. Make the rectangle's background color black.

   Keep your goal in mind. You want a long antenna-like curve on one side of your rounded rectangle.

2. Using the Border Radii controls (*Figure 6-3*), click the left top and left bottom buttons.

This makes the rounded rectangle flat on one side and rounded on the other.

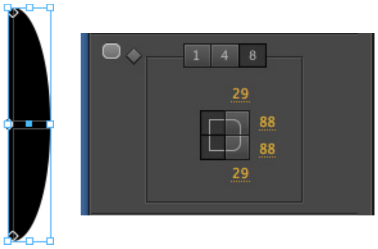


FIGURE 6-3

The Border Radii controls give you a rough idea of the shape your rounded rectangle will take. Here, the left buttons are pressed in, making that side of the element flat instead of curved.

3. Select the rectangle and press Ctrl+D (⌘-D).

   The duplicate rounded rectangle is directly above the original.

4. Change the background color of the duplicate to white.

   The assumption here is that the stage is white. If your stage is a different color, use that color for the duplicate rounded rectangle.

5. Drag the white element to the left revealing enough of the lower rounded rectangle to form an antenna as shown in *Figure 6-4*.

   If possible make sure that your antenna is symmetrical and slightly pointed on both ends.

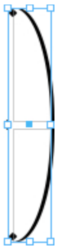


FIGURE 6-4

Edge doesn't give you an easy way to draw a simple curved line like this bug antenna. This was created with two rounded rectangles: one black and one white. The white one placed over the black creates a pseudo-mask so only a portion of the black rectangle is visible.

6. Select both the black and white rounded rectangle and then right-click (Control-click). Choose Convert to Symbol from the shortcut menu.

   The Create Symbol dialog box appears.

7. In the Symbol Name box, type *Antenna*.

   There's an instance of the newly created Antenna on the stage.

8. Attach two antennae to your roach.

   You can use the one on the stage and drag another antenna from the Library. At this point, your roach should look something like *Figure 6-5*.

TIP In cases where you want lines of uniform thickness, you may want to create a rectangle or rounded rectangle with a border. Then, set the background color to none. At that point, you can create a mask, as described in steps 3 to 5, to hide portions of the border. The method described here is better for creating lines with a more organic look.

Nesting Symbols within Symbols

So far your roach consists of three discreet symbols: Body, Leg, and Antenna. Each symbol is used more than once and they look like they're connected because they're placed on top of each other. As far as Edge is concerned, these symbols aren't really a single unit, they're just elements on the stage.

Edge lets you place symbols inside of other symbols. The technique is called *nesting,* and it's perfect for bug building. You want to be able to move your roach around the stage without having to reposition each leg and antenna individually. The solution? Select all the bug parts on the stage and then right-click (Control-click). A shortcut menu appears; choose "Convert to Symbol." When the Create Symbol dialog shows up, name your masterpiece *Roach,* and then click OK. Now that you have a Roach symbol in your Library, you can create as many as you want. As you see, the process is like the one used to create the legs.

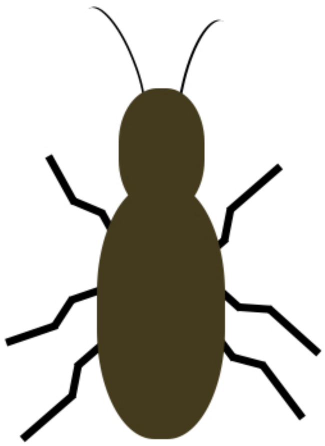


FIGURE 6-5

This roach is made from only three symbols. In the Library, they are named Body, Leg, and Antenna. Instances of symbols can be altered by changing their properties so they can perform multiple duties.

Working with Symbol Timelines

Your roach isn't intended to be a still life, so you need to add some action. Specifically, it needs leg action. Now that your roach is a symbol, you can drag multiple roaches from the Library and place them on the stage. Wouldn't it be great if each roach came with moving legs? With Edge symbols you can accomplish that. Animate once in the master roach, and all the instances that you add to the stage will have the same moving parts.

Each symbol has its own timeline, which is helpful in many ways. First of all, it cuts down on clutter in the main timeline. Every single element and property keyframe doesn't have to be in the main timeline Secondly, you can start and stop symbol timelines from playing. For example, you can use the symbol timeline to control whether the roach's legs are moving, as you'll see on page 127.

The first step is to animate the legs, the start and stop business will be handled later. Follow these steps:

1. In the Library, right-click (Control-click) the Roach symbol and choose Edit from the shortcut menu.

   The stage disappears, and the Roach remains on a white background: the Roach symbol is open for editing. In the upper-left corner you see the words: "Stage / Roach," as shown in *Figure 6-6*. When you have symbols inside of symbols, you can use these "breadcrumbs" to remember exactly where you are. You can also click them to close the symbol you're editing and jump up to a parent symbol. For example, if you click Stage, you close the Roach symbol and return to the stage and the main timeline.

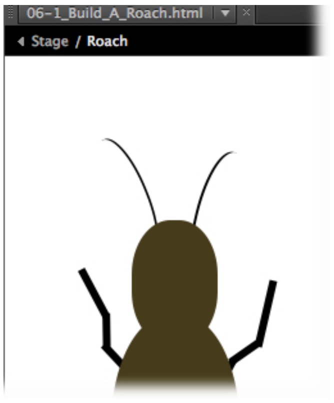


FIGURE 6-6

Right-click (Control-click) a symbol in the Library and choose Edit and you see an editing space devoted to that symbol. No other elements are visible. To return to the stage and main timeline, click the word Stage.

2. Go to the Elements panel and click the show/hide eyeball next to bugBody.

   The bugBody is now hidden. This gives you a full view of all the legs, making it easy to edit them.

3. Select one leg.

   In addition to the handles on the bounding box around the symbol, there's a blue square in the center. That's the Transform Origin. When you use the Rotate property, the symbol spins around this point.

4. With the Transform Tool (Q), drag the Transform Origin from the middle to the end of the leg that is usually hidden by the body as shown in *Figure 6-7*.

   You can't move the Transform Origin with the Select Tool; you have to use the Transform Tool. Once the Transform tool is selected, the Transform Origin changes to a crosshairs symbol as shown in *Figure 6-7*. It will be easier to position the legs and to create more realistic motion if the legs rotate around an end point rather than the center.

5. Move the Transform Origin for all the legs.

   Each leg symbol will be able to move independently around its Transform Origin. This might be a good time to reposition the legs a bit if you think it's necessary.

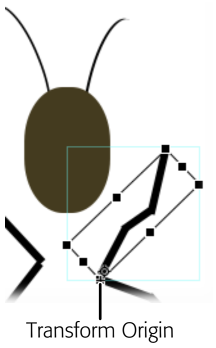


FIGURE 6-7

Here, bugBody is temporarily hidden. The Transform Origin has been moved to the left end of the selected leg. Now, when the leg is positioned using the Rotate property, it will rotate around that point.

   The emphasis should be on getting the end with the Transform Origin into just the right spot for the most realistic leg movement.

6. In Elements, click the eyeball next to bugBody.

   All parts of your roach are visible again. The legs are partially hidden by the bug's body.

7. Make sure the Auto Keyframe (A) is on and Instant Transitions are off. Then, create a starting pose for your roach.

Animation within a symbol is just like animation on the stage. Move the playhead to a position and then arrange the elements the way you want them at that point in time. To pose a leg, select it and then use Rotate in the Properties Panel to move it into position. Use this method to create a starting pose for each leg.

TIP To create the complete leg motion, you set several different leg positions in the symbol's timeline. You want the last position to match the first position. That way, if you loop the animation it will appear smooth. To remember the original position of the legs, mark those first positions with small rectangles, as shown in *Figure 6-8*. You can use these rectangle markers when you set the last leg position. When you're done, just delete the rectangle markers.

8. With the playhead at 0:00.000, select all the legs and then in Properties, click the diamond-shaped button next to Rotate.

   Edge creates Rotate property keyframes for each leg's starting pose.

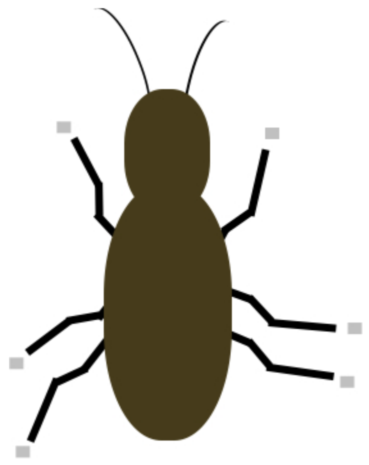


FIGURE 6-8

Temporary position markers are handy tools. Here, small rectangles mark the starting pose for each of the legs. That makes it easy to recreate the pose for the final frame of the animation.

9. Drag the playhead down the timeline to the 0:00.250 mark. Move all the legs to a new position.

   For skittery roach action, it's best if the legs don't all move in unison like marching soldiers. Front legs may be using a pulling motion while back legs push. The left rear and right rear legs may be moving in completely different directions. Use your best guess now; you can always fine-tune the leg motion later.

10. Create new leg positions for the 0:00.250, 0:00.500 and 0:00.750 marks. Lastly, move the playhead to the 0:01.000 mark and put the legs back in the starting position.

With the legs in the same position in 0:00.000 and 0:01.000, you'll be able to loop the animation smoothly. See the note on page 128 for help marking the leg position. Naturally, if you want faster leg action, shorten the time span. If you want slower action, use a longer time period for your four new leg poses.

11. With the playhead at the 0:01.000 mark, click the Add Trigger button on the timeline.

    The Trigger box opens, where you can add an action that takes place at this point in time.

12. On the right of the Trigger panel, click Play from.

    Edge adds a line of code that sends the playhead to the one second mark:

    ```
    sym.play(1000);
    ```

13. Replace the 1000 with 0.

    The zero moves the playhead to the beginning of the timeline.

14. In the upper left corner of the workspace, click the word Stage.

    The Roach symbol is closed. You're no longer editing it. The workspace you see is the stage. Your animated roach rests peacefully on the stage.

15. Press Ctrl+S (⌘-S) to save your project. Then press Ctrl+Enter (⌘-Return).

    Your animation opens in your web browser for previewing. You should see some good leg action even though your roach isn't moving anywhere. On page 188, you'll learn how to start and stop the roach timeline. That's a job for JavaScript/jQuery code.

Just to recap what's taken place so far in the Build-A-Roach project. You drew three elements: Body, Leg, and Antenna. You turned each into a symbol and then used multiple instances to build your roach. Then you selected all the parts and turned that into your master roach symbol. Lastly, you used the timeline in the roach symbol to animate the legs.

- You can turn any element or group of elements into a symbol using the Modify→Convert to Symbol (Ctrl+Y or ⌘-Y) command.
- You can modify an existing symbol by right-clicking (Control-clicking) its name in the Library panel and then choosing Edit.
- You can place symbols inside of other symbols, a technique called *nesting*. (Perhaps not the most pleasant term when you're discussing roaches.)
- Symbols have their own timelines, so you can animate the elements within a symbol.

Animating a Symbol on the Stage

To create a respectable roach, you not only need to have legs that move, you need to move the roach around the stage. In the previous exercise, you opened the Roach symbol for editing and then used its timeline to animate the legs. Now that the legs move back and forth, you'll want to make the roaches move around the screen. That's a job for the stage and the main timeline. If you've been working through the exercises so far in this book, that's a cinch. It's the same old animation two-step. Move the playhead to a point on the timeline, and then position the element where you want it at that point in time.

Here are some tips and suggestions for creating a satisfying roach animation:

- Think about the route that you want your roach to travel. If you intend to loop your animation, you want your roach to circle back around to the starting position.
- Two or three seconds should be enough time to scurry around an average size stage of 550x400 pixels.
- Make sure the Auto Keyframe (A) is on and Instant Transitions (I) is off before you reposition your bug.
- Use the Location and Rotate properties to move and position your roach. Consider using some erratic movements both in direction and speed.
- To view leg motion while you position your roach and work with the Timeline, select the symbol, then turn on Properties→Playback Actions→Scrub.
- To create a multi-roach animation, just click on Roach in the Library and drag another instance onto the stage. Use the scale properties to make roaches in different sizes.

NOTE Want to compare your Build-A-Roach project to some finished code? Get *06-1_Build_A_Roach_done* from the Missing CD at *www.missingmanuals.com/cds/edgepv5mm*.

Duplicating and Renaming Symbols

Chances are you'll put in a lot of work when you create a complex symbol. Suppose you're creating a street scene with several different cars. You don't want your cars to all look alike. So, you create your first car: a Prius. Then you right-click (Control-click) the car name in the Library and choose Duplicate. A new item appears in the list of symbols: Prius Copy 1. This car is a new, separate symbol; right now it just happens to look like a Prius. That can change when you open and edit the symbol. Right-click (Control-click) and choose Edit from the shortcut menu. Perhaps you turn it into a Chevy Volt. It's still work, but it's a lot easier making changes to an existing car than creating a new one from scratch. When you're done editing, you'll want to change the name to match the new vehicle. So…you guessed it. Right-click (Control-click) the name in the Library and choose Rename from that same shortcut menu.

Deleting or Undoing a Symbol

There may come a time when you want to delete a symbol that you created. If you no longer need the symbol and want to get rid of all the instances of that symbol, it's easy. Just right-click (Control-click) the symbol name in the Library and choose Delete. A warning appears that reminds you that you have instances of the symbol on the stage and that asks: "Are you sure you want to delete the selected symbol?" Say yes, and the symbol is removed from the Library, and all the instances on the stage are deleted.

NOTE If you accidentally delete a symbol and all the instances of that symbol in your project. You can always bring it back with an Undo command (Ctrl+Z or ⌘-Z).

What if you want to keep the elements inside, but remove the symbol definition? For example you want to break a symbol back into the elements it was made from. That requires a couple of extra steps:

1. Open the symbol for editing by right-clicking (Control-clicking) on its name in the Library.
2. Choose Edit from the shortcut menu.
3. Press Ctrl+A (⌘-A) to select all the Elements in the symbol.
4. In the upper left corner of the workspace, click the word Stage to close the symbol and return to the stage.
5. Choose Edit→Paste to paste the elements onto the stage. When that's done, they aren't part of a symbol. They're just individual elements.
6. Right-click (Control-click) the symbol name in the Library and choose delete. The instances of the symbol will be removed from the stage, but the elements you copied will remain.

Create a Button Symbol with Rollover Action

One of the most common tools on the web is the lowly button. Many buttons include some sort of *rollover* action. That is, when someone moves her mouse over the button, the button changes to indicate it's clickable. When she clicks the button, it may change apperance again. If you're developing a navigation system for a website, you usually want some uniformity when it comes to buttons. That's a perfect job for a symbol. You can use the symbol's timeline to create different looks for normal and mouseover states.

1. Open and save a new Edge project with the folder name and filename *Button*.

   Make sure you save your project in an empty folder.

2. With the Rounded Rectangle tool, draw a button-sized rectangle.

   You don't have to worry about making adjustments now. You can make changes after you've converted the rectangle to a symbol.

3. Right-click (Control-click) the rectangle and choose "Convert to Symbol" from the shortcut menu, as shown in *Figure 6-9*.

   A box appears prompting you to name the new symbol.

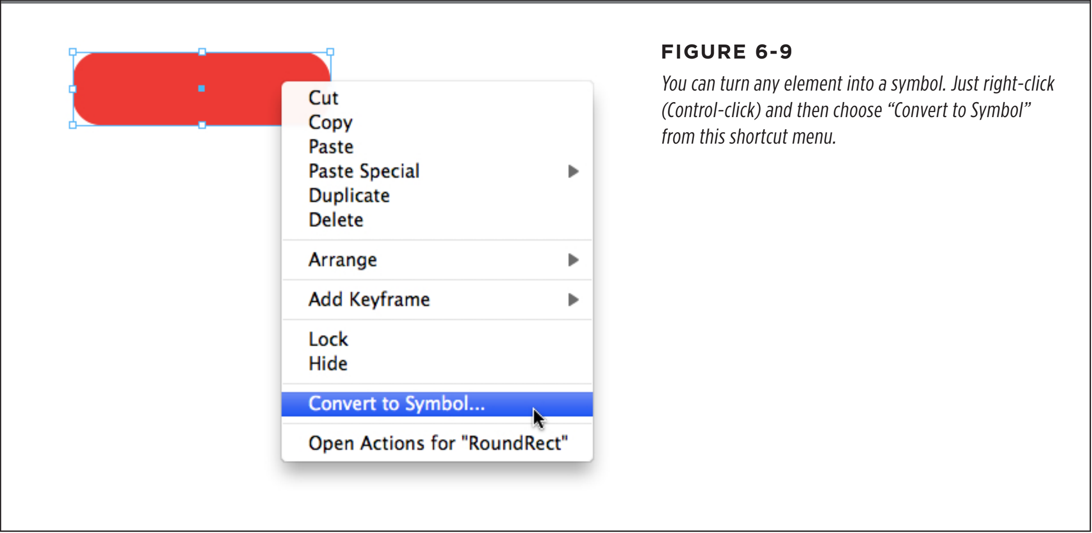


FIGURE 6-9

You can turn any element into a symbol. Just right-click (Control-click) and then choose "Convert to Symbol" from this shortcut menu.

4. Type the name *Button* and click OK.

   You've transformed the simple rectangle into a symbol. Look in the Library panel under Symbols→Library. If it's not already open, choose Window→Library and click the triangle button next to Symbols. The Library panel opens with three expandable lists: Assets, Symbols, and Fonts. Button is in the Symbols list.

5. Double-click the unmodified instance of btnHome on the stage.

   The stage and all the other elements darken except for the unmodified instance of Button, which is available for editing. This technique is called "editing a symbol in place." It gives you the opportunity to see your symbol's relationship with other stage elements while you make changes. The Timeline displayed when you're editing a symbol is the Timeline for that symbol.

6. With the playhead at 0:00, press Ctrl+L (⌘-L) to create a label named *normal*. Create a second label at 0:01 named *over*.

   You'll use these labels, shown in *Figure 6-10*, to control the color of the button.

7. With the playhead at 0:00, choose a background color for the button's normal state. Then, in Properties, click the Add Keyframe button next to Background Color.

   The button now has different colors at different points of the timeline.

8. Right-click (Control-click) the rounded rectangle and choose Open Actions for "RoundRect" from the shortcut menu. Choose the *mouseover* trigger.

   The Actions panel appears, ready for you to add actions to the mouseover event.

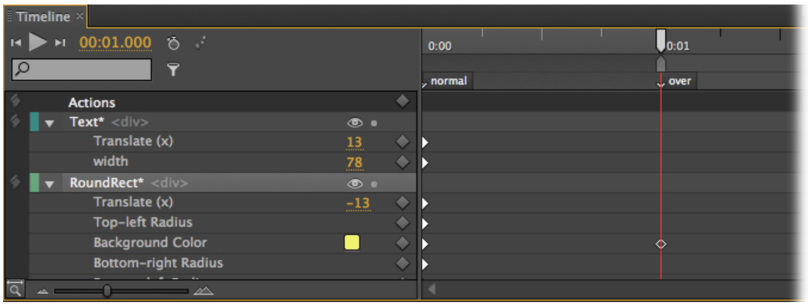


FIGURE 6-10

Each symbol has a Timeline all its own. Here the button has labels in its Timeline to mark the "normal" and "over" state of the button.

9. On the right of the Actions panel, click "Stop at."

   The code to go to and stop at a point on the Timeline appears in the actions.

10. Replace the argument in the *stop()* function that reads "1000" with *'over'*.

    Now the line of code reads:

```
sym.stop('over');
```

    Don't forget to include the quotes when you use a label as the argument for a function.

11. In the upper-right corner of the Actions panel, click the + button. Then choose *mouseout* from the list of triggers.

    A new tab appears in the Actions panel, ready for your mouseout code.

12. On the left side of the Actions panel, click "Stop at."

    The code to go to and stop at a point on the Timeline appears in the actions.

13. Replace the argument in the stop() function that reads "1000" with *'normal'*.

    Now the line of code reads:

```
sym.stop('normal');
```

14. While your symbol is still open for editing, in Properties, deselect the Autoplay box.

    This step prevents your button from automatically playing the Timeline. (If it did so, the button would flash both colors whether the mouse was over it or not.)

15. In the upper-left corner of the work stage window, click Stage.

    You leave the symbol editing space and return to the stage. Now the Timeline displayed is the main Timeline for your project.

16. Save (Ctrl+S or ⌘-S) your project. Then press Ctrl+Enter (⌘-Return) to preview it.

    Your button behaves as advertised. Initially, it shows the normal state. When the mouse is over the button, it changes color.

Most buttons have some sort of label that identifies the button and the expected action when it's clicked. For example, you may drag an instance of button out of the Library and then place text over it to create, say, a Home button or a Contact Us button. There's one slight problem with this method, though. When the mouse is over the text box, the button doesn't change color. That's because the text box is over the button, blocking its rollover behavior. There are a few different solutions to this conundrum. In Chapter 8, you'll see how to fix it with JavaScript/jQuery code. For now, you'll learn a graphic workaround, which is to place the text beneath the button and then dial down the Opacity of the button. The color is softer but at least the text shows through; see *Figure 6-11*.

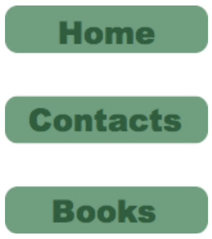


FIGURE 6-11

These buttons were created with a single Button symbol using a dark shade of green and a lighter green for the rollover. The text labels were placed beneath the symbols. Then the button's opacity was set at 75 percent so the text shows through.

In this example you saw how you can change the color of a button using a symbol's independent Timeline, labels, and a couple of "Stop at" actions. If you'd like to have snazzier animated buttons, you could use "Play from" actions. On *mouseover* and *mouseout*, the symbols' Timelines would play a section of the Timeline. You'd need to have some sort of triggers at the end of each sequence to either stop the Timeline from playing or to move the playhead back to the "normal" state.

Making Your Button Open a Web Page

If you followed the previous exercise, you now have a button that behaves kind of like a button. If this were a button on a web page that displays a home page, you could add a trigger and an action to do that. You could add that behavior inside the symbol, but it's easier to do it right on the stage. Suppose you have an instance of the button stymbol on the stage called btnBooks, here are the steps to add an action that opens a web page.

1. Right-click (Control-click) the instance of btnBooks on the stage, and choose Open Actions for "btnBooks." Then choose the *click* trigger.
2. On the left of the Actions panel, click the Open URL button.
3. Edit the *window.open()* function to use the desired web address.

When you're done, you should have a line of code that looks something like this:

```
window.open("http://missingmanuals.com", "_self");
```

At this point, it's pretty clear that you can make a series of animated buttons that could serve as the navigation system for a website. You don't have to make each button from scratch, either. A single rollover button symbol is reusable. All you need to do to create distinct buttons is to add text for a label. For an example of the button described here, see *06-2_Button_Symbol* from the Missing CD at *www.missingmanuals.com/cds/edgepv5mm*.

Building a Drop-Down Menu System

Anyone who's built a website knows that the process is like building a brick wall. You start off by putting a couple of bricks down. Add some mortar and then add some more bricks. You can use that same technique to build a drop down menu out of rollover buttons (*Figure 6-12*). The great thing about drop-down menus is that anyone who's used Windows or a Mac knows exactly what to do.

Think of your Button symbol as the brick you use for construction. You create it just as described back on page 131. Your button has two states: normal and over. When you place it on the stage, you dial down the Opacity to something between 20 and 35 percent. You place the label text underneath the semitransparent button. You can create as many buttons as you need for your drop-down menu. Just add more bricks. Use click triggers to show and hide the buttons, just like the File menu of your favorite word processor shows and hides commands. Using Edge actions, those buttons can lead to a new web page as described in the previous section. Or they can display a different part of the timeline as done in the example: *06-3_Dropdown_Menu_done* from the Missing CD at *www.missingmanuals.com/cds/edgepv5mm.*

TIP Chapter 8 provides details on controlling elements nested within symbols using JavaScript and jQuery. Those skills provide even better ways to show and hide the buttons in a drop-down menu.

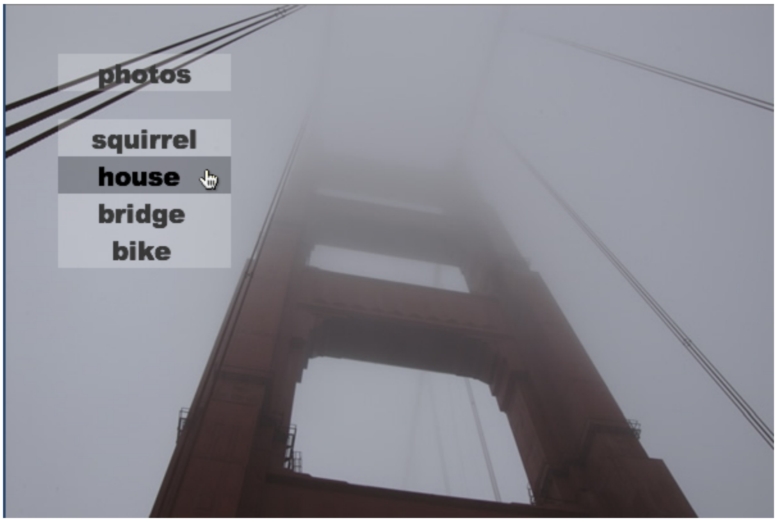


FIGURE 6-12

The buttons on this drop-down menu display different pictures. Click the Photos button at the top to show and hide the submenu. As explained on page 135, they could just as easily open a new web page.

Working with Basic HTML and CSS

When you build a project, Edge automatically creates several different files. The Hypertext Markup Language file (.html) is the stuff web pages are made of. Web browsers read HTML files and display the results as web pages. The Cascading Style Sheet file (.css) supports HTML documents, providing formatting and positioning instructions. You don't have to be an HTML/CSS wizard to work with Edge. On the other hand, when you understand how HTML and CSS documents work, you're in a better position to customize your projects to meet your needs. Edge compositions live within the HTML/CSS environment, and your audience views your work in their web browsers. In some cases, the files that Edge produces will be all you'll need. In other instances, your Edge composition may be just a part of a larger page. For example, your project may be a banner ad. Knowledge of HTML and CSS helps you when it comes to sizing and positioning your project.

In this chapter, you'll learn how to read and interpret HTML and CSS documents. Along the way, you'll see how HTML's IDs and classes are used to identify elements on the web page. Toward the end of the chapter, you'll learn how to open and animate elements in existing web pages and how to position your Edge composition in another web page.

Reading HTML Documents

It's not hard to learn how to read HTML. If you've spent any quality time developing web pages, you likely have some skills. You can open an HTML document in any simple text editor like Notepad (Windows) or TextEdit (Mac). Nearly any web browser lets you view the source of the current web page. Just right-click (Control-click) on the page and choose View Source from the shortcut menu. The code behind complex web pages can be stunningly long and cryptic. However, once you know the basics, you can tackle a page chunk by chunk. With some practice, you can get a good idea of what's going on behind the scenes.

At its simplest, a complete "Hello World" web page might look like this:

```
<!DOCTYPE html>
<html>
<head>
    <title>Hello World</title>
</head>
<body style="margin:10;">
    Hello World
</body>
</html>
```

Web pages are made up of different elements, sometimes referred to as nodes. Like all web pages, this document has a *head* element and a *body* element. Often the head holds special details that don't appear on the page. For example, the head is likely to hold style information that is applied to content in the body. The head may also hold script tags for JavaScript code that performs some sort of automation or animation magic. Savvy web designers use special tags in the head portion of a page to provide a description of the page and a list of keywords. These details may be used by search engines to categorize the page. The head may include JavaScript code to automate elements on a web page or, as you'll see later, the head may hold page formatting instructions. The body holds the actual content that's displayed on the web page. This is where text, pictures, and your Edge compositions appear.

Playing Tag the HTML Way

If you want words to appear on a web page, all you need to do is place your text in the body portion of the document. For example, see the "Hello World" message on line 7? For this simplest of HTML documents, those words are the only thing that would appear in the document window. So what's with those brackets and all that other stuff? Those are tags. They don't show up on the web page, but they provide necessary information to web browsers. Tags are differentiated from content by their angle brackets (< >). Tags usually come in pairs, like the ones on line 4:

```
<title>Hello World</title>
```

These tags give the web page a title, and most browsers stick that title on the tab or title bar for the web page. The first tag lets the browser know that what follows is the title for this web page. The second tag is identical, except for its slash mark (/). That tag says to the browser, "OK, that's the end of the title." As you can see from a quick look at lines 3 and 5, it's common for one element to be completely inside another. In this case, the <title> element is a "child" of the <head> element, which in turn is a child of the <html> element.

Occasionally, you'll come across a single tag. For example, the tag for a line break in text is <br/>. Notice that in this case the slash comes right before the last bracket.

TIP A big part of being able to make sense of HTML code is knowing all the official tags and what they do. If you're looking for a cheat sheet, pick up *HTML & XHTML Pocket Reference* by Jennifer Niederst Robbins (O'Reilly). If you want a complete manual for learning the latest, greatest version of HTML, turn to *HTML5: The Missing Manual* by Matthew MacDonald (O'Reilly).

The first line of the code on page 138 is a single tag but an important one. It tells browsers what type of a document follows. That way the browser knows how to interpret all those tags. HTML is certainly the most popular DOCTYPE these days, but you might also come across xml or xhtml. Sometimes you'll see a version number displayed as in HTML 4.0 and other details. These generally aren't necessary for your Edge projects. Previous versions of HTML and XHTML used more complicated DOCTYPE details, but with HTML5 a simple "HTML" is all that you need.

The last thing to examine in the "Hello World" code is line 6. Here the <body> tag includes the *attribute*: style="margin:10;". In this case, the attribute is named style and it sets the margin for the body of the web page. Attributes appear within the brackets of opening tags. The attribute name is followed by the assignment operator (=), which then assigns values to the attribute. Values appear inside quotes, which may be either double (") or single ('). In Edge, when you set the property for an element, often behind the scenes, Edge is setting the value for an attribute. The width and height of an image, the color and style of text—these are all attributes of specific elements.

TIP Edge uses HTML tags, too. Select an element, and next to the name, you see the related tag. Often, you'll see the generic but versatile <div> tag. For a list of the common tags seen in Edge, turn to page 69.

Creating a Hyperlink with HTML

If there's one thing that's made HTML king of the World Wide Web, it's the hyperlink. Hyperlinks are the threads that form that web. Click a linked word, and suddenly you're in a different part of the universe (or at least, the Web). You create the links using the <a> anchor tags. Here's an example of a hyperlink:

```
<a href="http://www.stutzbearcat.net">click me</a>
```

Like most HTML tags, the anchor tag comes in pairs: *<a>in between stuff</a>*. In this case, the *href* attribute provides a web address. The *href* stands for hypertext reference; the equals sign assigns a value to the reference inside double quotes. The specific value here is a web address. The words in between the two <a> tags, "click me," are visible in the browser window. Depending on the tag's formatting, they may appear underlined or highlighted in some way to show that they're a link.

```
<font face="Cooper Black" size="3"> Home of the <em>legendary</em>
<strong>Stutz Bearcat</strong></font>
```

When a web browser like Chrome, Safari, Firefox, or Internet Explorer reads this text, it knows how to display the text in the browser window. For most browsers, the <em> tags indicate italics. Think emphasize. The <strong> tags specify bold text. The browser applies the formatting instructions and shows the text. It displays the proper typeface if it's available; otherwise it uses a substitute font. So HTML coding works fine from the audience's point of view. For designers, it can be a bit of a pain. One of the problems of HTML coding is that the message gets a bit lost in the formatting tags. Complicated HTML coding is hard for human eyes to read, and that makes it easy to foul up. When you want to make changes, it's a lot of work to go in there and tweak all those bits of embedded code. The solution? CSS to the rescue.

Reading CSS Files

CSS is the acronym for *Cascading Style Sheets*—an ingenious system for formatting HTML text. As the Web has matured, CSS has taken on more and more responsibilities. In HTML5, the role of CSS is even greater. If you want to read up on how CSS works, you can get an excellent introduction in David Sawyer McFarland's *CSS: The Missing Manual* (O'Reilly). A basic understanding of CSS will help you when you're working with Edge. Edge uses CSS behind the scenes to format and position elements. This book provides a quick overview of CSS and an example or two to get you started.

Formatting and style information can be stored in three different locations, as explained in the box on page 142. As the Web grew and pages became more sophisticated, new and better methods were developed to dress up those pages. These days, the fashionable technique is to use a separate Cascading Style Sheet (.css document) to format web pages. A line in the head of the HTML document links the stylesheet definitions to the web page. It might look like this:

```
<link rel="stylesheet" href="barebones_edge.css" />
```

The link tag is a single tag; there's no corresponding end tag. The link tag is used to link more than just CSS files, so the *rel* (relationship) attribute is used to define the linked file as a stylesheet. The *href* attribute points to the CSS file which, in this case, is named *barebones_edge.css*. There's no path in this case, because the CSS file is in the same folder as the HTML file. If the CSS document is in a different folder, a standard path description is needed with the filename. For example, if there's a special folder that holds all the CSS files, the complete path and name might be *css/barebones_edge.css*.

The underlying philosophy is that it's best to separate the formatting from the content. You create styles (type specs) for body text, major headlines, subheads, captions, and so forth. You store those definitions in a style sheet. Then, in the text, you tag different portions, indicating the styles they should use. In effect, you say: "This is a major headline, this is body text, and this is a caption." When the browser goes to display your web page, it comes to the text tagged as a headline, and then it looks up the type specs in the style sheet. It does the same thing for the body text and the captions. From a designer's point of view, this system is a tremendous timesaver. If you want to change the caption style for a website that has 400 captioned pictures, all you need to do is edit one definition in the style sheet. If all those type specs were embedded HTML code, you'd need to make 400 separate edits.

CSS style sheets are a little like those wooden Russian dolls where each one is nested inside another. Starting from the outside and working in, here's what you find in a CSS style sheet. A style sheet is a list of formatting specifications. Each formatting spec has a selector that identifies the HTML tag that it formats. That tag could be the paragraph tag <p>, the heading tag <h1>, or an anchor tag <a>. In CSS lingo, the formatting spec is called a *declaration block*. The declaration block is contained inside curly braces {}. Within those curly braces are specific declarations that define fonts, styles, colors, sizes, and all the other properties that can be defined in CSS. The declarations have two parts: a property and a value. So in CSS, if that property is *font-size*, then the value is a number representing point size. A CSS definition to format an <h1> heading tag might look like this:

```
h1 {
font-family: Arial;
font-size:18px;
font-weight: bold;
color: red;
}
```

The first line has the selector for *h1* headings, usually the biggest, boldest heading on a web page. The next four lines show pairs of properties and values. On the left side of the colon is the property, which is hyphenated if it's made up of more than one word. On the right side is the value assigned to that property.

Applying CSS Styles to Element IDs and Classes

You've seen how you can connect CSS styles to particular tags, like the heading1 tag <h1> or a paragraph tag <p>. Those formatting specs affect all of the headings and paragraphs that use the tag. But what if you only want to change the appearance of a specific heading or paragraph? That's the job for an ID or a class. You can give any element on the page an ID. That's exactly what Edge does behind the scenes when you name an element. Use IDs for elements that appear only once on the page. If you create a *LeftSidebar* ID, you can have only one *LeftSidebar* on the page. That's just how IDs work. If you need to apply a style to more than one element on the page, you can create a *class*. You can apply a single class name to several elements. For example, perhaps you're creating pull quote boxes (teasers that boldly display a line or two of text pulled from the main body of your document). So you specify a big, bold font and create text boxes that float on the right side of the page. Now you can create a *PullQuote* class and apply that to any chunk of text that you want to display those properties.

You assign a specific ID or class to an element through a tag attribute. For example, maybe you'd like to give a sales pitch its own ID. In the body of your HTML document, it might look like this:

```
<p id="pitch">What can I do to put you in a Stutz Bearcat today?</p>
```

Then in your CSS file, there'd be a definition for the *pitch* ID:

```
#pitch {
font-family: Arial;
font-size:18px;
font-weight: bold;
color: red;
}
```

POWER USERS' CLINIC

Three Places for Style Specs

There's more than one place to provide style and formatting details when you build a web page.

- **Inline styles.** As shown on page 140, formatting and style details can be interspersed with the HTML code and web page content. This method is considered old school and not used much today, except for an occasional bold or italic markup. It's also helpful for those cases where a particular element requires some exceptional formatting that isn't part of the template or previously defined styles.
- **Internal style sheet.** CSS styles can be listed in the head of the HTML document. In this case, the style specifications are separate from the content that they format. The listed style definitions are used only in the document in which they appear. This method is fine for standalone or unique pages.
- **External style sheet.** Styles can be stored in a separate document with a filename that ends in *.css*. This external style sheet contains only style definitions and no content. If it's helpful, the file may also include comments. A single style sheet can be applied to many HTML documents. This is great when you have a big website with dozens of pages, many of which have a similar appearance. Also useful, a single document can have more than one stylesheet applied to it.

When you build an Edge project, Edge creates CSS files to format and position the elements. In Preview 4 and 5, these details are hidden from you; in earlier versions, Edge created an external CSS stylesheet, with a comment at the beginning of that document warning you not to edit the file. HTML permits the use of multiple stylesheets for a single page, so, if necessary, you can create another external CSS file to format other parts of your web page.

What happens when the same element has conflicting styles? Say a paragraph <p> element has an inline style applied to it, but there are also styles for <p> in an internal or external style sheet? CSS is designed so that the styles closest to the content take precedence. So an inline style will override the styles listed in the head of document , and that internal style sheet overrides an external style sheet saved as a separate file. This is a good method because it makes it easy to override a style occasionally. However, if your pages aren't behaving as expected, it's worth double-checking for CSS conflicts.

The pound sign (#) is what distinguishes an ID from any other tag. The process for defining a class is similar. If you want to apply a class named *pause* to a paragraph of text, it would look like this:

```
<p class="pause">Hang on a second. Let me check that price with my manager.
</p>
```

Then, in your CSS file, there'd be a definition for the *pause* class. And of course, that class might be applied to more than one paragraph.

```
.pause {
font-family: Times;
font-size:12px;
font-weight: bold;
color: black;
}
```

The cascading aspect of CSS gives designers a double dose of formatting power. Suppose your web page has two main sections: the body and a sidebar. The body has a white background color, and the sidebar's background is a dark blue to set it off from the body. You're going to need a different font color for those two places. CSS has a solution. In effect, you can say, "When a paragraph is in the body, make the font color black. When a paragraph is in the sidebar, make the font color white." You create those instructions in your style sheet, like this:

```
body p {
font-family: Times;
font-size:12px;
font-weight: bold;
color: black;
}

#sidebar p {
font-family: Times;
font-size:12px;
font-weight: bold;
color: white;
}
```

In this way, that simple paragraph <p> tag takes on different characteristics depending on where it's located.

Reading the HTML Edge Creates

When you're in Edge and you choose File→Save, Edge creates several files, including an HTML file that describes a web page with your composition. You can open that HTML file and read the contents. You may be surprised to see that there's actually not a lot of code inside. For the most part, it's made up of links to other files that make your composition work. Here's the HTML from a bare-bones Edge project:

```
<html>
<head>
<meta http-equiv='Content-Type' content='text/html; charset=utf-8'>
<title>Bare Bones</title>
<!--Adobe Edge Runtime-->
<script type="text/javascript" charset="utf-8"
        src="01-1_Bare_Bones_done_edgePreload.js"></script>
<!--Adobe Edge Runtime End-->
</head>
<body style="margin:0;padding:0;">
<div id="stage" class="EDGE-170190213">
</div>
</body>
</html>
```

The first line declares that it is an HTML5 document. Everything else is inside the two <html> tags. At the next level, the page has the two standard elements, <head> and <body>. The <head> includes a <title>. In this case, the text reads *Bare Bones*. The rest of the head fits in between two HTML comment tags: The first comment reads <!--Adobe Edge Runtime-->. The last reads <!--Adobe Edge Runtime End-->. Between these comments are statements that link a JavaScript (.js) file using the <script> tag. As you might guess from the word "Preload" in the name, this JavaScript code loads resources that Edge needs. Specifically, the preloader identifies other JavaScript files and libraries, some of which are in the *edge_includes* folder. Edge created all these files when you performed that Save command, and in many cases you don't have to mess with them. You won't need to change the links in this HTML document, unless you do something like move the files. And in most cases, you won't edit the external files. In fact, keep your hands off the four JavaScript files in the *edge_includes* folder. They contain the code that makes Edge compositions do their magic. They have names like:

- jquery-1.6.2.min.js
- jquery.easing.1.3.js
- edge.0.1.4.min.js
- yephope.js

On the other hand, once you become proficient in JavaScript/jQuery, you may find yourself making changes to the JavaScript files that are in the same folder as your HTML document. If you save a project named *Bare_Bones*, you'll find these JavaScript files along with your .html web page and .edge project files:

- Bare_Bones_edge.js
- Bare_Bones_edgeActions.js
- Bare_Bones_edgePreload.js

You'll learn more about tweaking these guys in Chapter 8 and 9. In general, Edge is pretty good about placing warnings at the beginning of files you shouldn't edit and providing tips for making changes to the others.

NOTE The box on page 98 describes two types of comments shared by JavaScript and other programming languages like C. The <!-- and --> tags are used by HTML. These particular tags are useful with JavaScript. They provide a way to deliver JavaScript commands to browsers ready to handle JavaScript. If a browser isn't able to handle JavaScript, the commands are ignored—treated like comments.

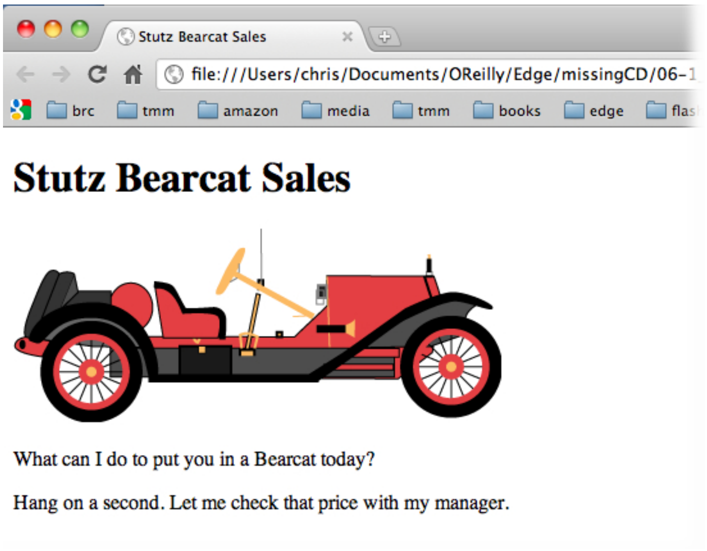
Opening an HTML Document in Edge

Want to apply some Edge magic to a web page that already has elements in place? You can do that by opening an existing document in Edge. For example, consider this web page, which isn't all that much more complex than the original "Hello World" example.

```
<!DOCTYPE html>
<html>
<head>
<title>Stutz Bearcat Sales</title>
</head>
<body>
<h1 id="headline">Stutz Bearcat Sales</h1>
<p><img src="images/StutzBearcat.png" alt="The amazing Stutz Bearcat
automobile" name="bearcat" id="bearcat" /></p>
<p id="pitch">What can I do to put you in a Bearcat today?</p>
<p id="pause">Hang on a second. Let me check that price with my manager.</p>
</body>
</html>
```

This web page has three bits of text. One has a heading <h1> tag. The other two have paragraph <p> tags. There's also one image on the page with a source .png graphic identified. All the elements, both text and image, have IDs applied. You don't have to do this, but it does make it easier to identify the elements inside Edge. As of this writing, Edge doesn't provide a way to add classes or IDs to elements once you've opened the HTML in Edge. In Edge, content that you create using Edge tools is called managed content. Content that is created outside of Edge is unmanaged content. In the Elements panel, managed content has an asterisk after the name. When you point to a managed element in the Elements panel, a tooltip lists the elements name (ID), tag, and in parentheses shows the word "managed."

Open the page in a browser, and it looks like *Figure 7-1*.



Stutz Bearcat Sales

What can I do to put you in a Bearcat today?

Hang on a second. Let me check that price with my manager.

FIGURE 7-1

Here's the simple web page that matches the code above. It consists of three blocks of text and the image of a snazzy auto. You can load this page in Edge and animate each of the elements.

If you want to experiment with opening an HTML document in Edge, get the Missing CD file *07-1_Load_HTML.html*.

1. In Edge, open *07-1_Load_HTML.html*.

   The Elements panel shows the elements existing in the HTML document (*Figure 7-2*), listing their IDs and tags.

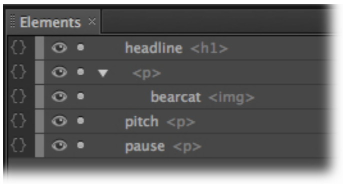


FIGURE 7-2

Give IDs to the elements in your HTML document when you create it in your favorite web-building tool. Then when you open that HTML in Edge, you see the IDs as element names. Here, headline, bearcat, pitch, and pause are all element IDs.

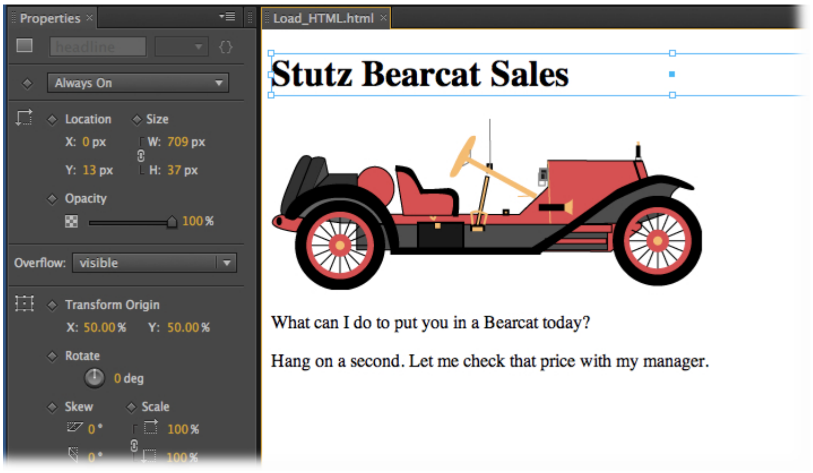
2. Click the car on the stage.

   The Properties panel shows the image's properties such as Width, Height, and Opacity. The car's HTML ID (bearcat) appears at the top of the Properties, but it is grayed out. You're unable to rename an unmanaged element (that is, change its ID). You're also unable to change the element's tag. You can do those things only with elements that you've created inside Edge.

3. In the Elements panel, click "headline."

   The Properties panel shows the same properties that were available for the image, as shown in *Figure 7-3*. However, the properties that you'd expect to see in a text box are missing. No Font, Size, or Color properties are available. In fact, you can't even change the text in the headline or the two paragraphs. If you need access to those properties in Edge, then you need to create text boxes inside Edge. If you're familiar with JavaScript, you can write your own code to make changes to elements that aren't managed by Edge. That process is described in the next chapter.

TIP If you change your HTML file in a separate editor while it's open in Edge, you'll be prompted to reload the page in Edge. Most of the time, you'll want to do that. But beware, you'll lose any unsaved changes or animation you've applied to the project.



Stutz Bearcat Sales

What can I do to put you in a Bearcat today?

Hang on a second. Let me check that price with my manager.

FIGURE 7-3

Edge gives you access to only a limited number of properties for existing elements on web pages that you open. Here the headline is selected, but there's no access to font specs.

4. Animate the car.

   You may want to start by making the car smaller; about 200 px wide works well. Then create a transition that moves the car across the stage. (See page 80 if you need to review the techniques to create a motion transition.)

5. Animate the headline and paragraphs.

   Perhaps you want the headline to drop in from the top of the page. The two paragraphs are hidden at the beginning of the animation. Then the pitch appears first: "What can I do to put you in a Bearcat today?" Shortly, the pause paragraph appears saying: "Hang on a second. Let me check that price with my manager."

You can use many of the Edge animation tricks you've learned on elements in a pre-existing document. You just need to work with the properties that are exposed in Edge. If you want to provide interaction, you can use triggers and actions, as described in Chapter 5. For an example, check out *07-2_Load_HTML_done*.

You save your Edge-altered HTML file as you would any Edge project. When you do, Edge creates the usual suspects: JavaScript files. If you take a peek inside the HTML file after you've saved it in Edge, you'll see that it didn't change very much. Edge adds the usual links to external files mentioned in the box on page 159. The body of the document remains as it was originally.

TIP As of this writing, it wasn't possible to add a link to a few words inside a text box created with the Text tool, but without digging into JavaScript/jQuery code. You can add a link to the entire text box, but not specific words within. One workaround for this is to create text in an HTML file like the example above. Make your links before you open the HTML in Edge. The only drawback is that you will not be able to edit the text inside Edge. For the JavaScript/jQuery way to create a link within text, see the box on page 187.

Placing Your Composition in an HTML Document

You won't always be content to create entire web pages inside Edge. For example, suppose you want to create a banner ad in Edge. You'll want to place the ad on a web page that has other content. The banner ad is just one element on the web page. Horizontal banner ads usually appear at the top of the page. They may need to be centered or float to the left or the right. If you're in the mood to experiment, you can use *07-3_Banner_Ad* from the Missing CD.

Here's the process. First you create your banner ad in Edge in the normal manner. If you're making a horizontal banner like the one in *Figure 7-4*, the stage dimensions might be something like 468x60 pixels. When you save your project, Edge creates all the usual files. You'll want to copy all the files and folders in your Edge project to the folder on your website that holds HTML files.

In your favorite HTML editor, open the page where you want to place your Edge composition. In another window of your HTML editor, open the page that you created in Edge. What you're going to do next is really just a simple cut-and-paste operation. In the Edge file, find and copy the code that launches the preloader.

As explained on page 144, that code is in the head of the HTML file that Edge creates, and it is between two comment tags that look like this:

```
<!--Adobe Edge Runtime-->
```

and

```
<!--Adobe Edge Runtime End-->
```

For example, if your project was saved with the name Bare_Bones, the complete code that you need to copy looks like this:

```
<!--Adobe Edge Runtime-->

<script type="text/javascript" charset="utf-8" src="Bare_Bones_edgePreload.
js"></script>

<!--Adobe Edge Runtime End-->
```

Copy the comments and everything that's in between them, and paste the lot into the head of your target web page.

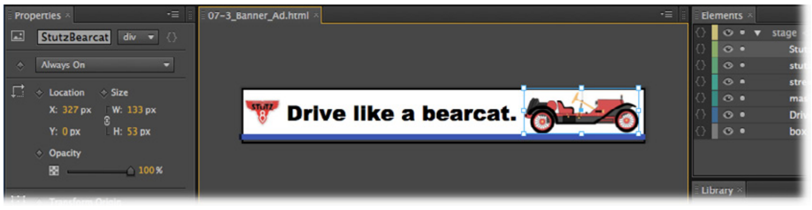


FIGURE 7-4
Edge projects like this banner ad don't usually stand alone on a web page. You're likely to create Edge projects that end up sharing space on a web page with other elements.

NOTE Versions of Edge before Preview 4 didn't use a preloader. In place of the preloader reference, you'd see several lines of code. Still, the procedure is the same: copy the two comment tags and everything in between.

Next, in the body of the HTML file that Edge created, find the <div> tag with the *stage* ID. It looks something like this:

```
<div id="stage" class="EDGE-204647159">
    </div>
```

Again, copy the tags and everything in between. This time, paste the code into the body of your web page. Close the Edge HTML file, and you can save or continue editing the web page. If you open your HTML page now, you'll find that the ad shows up where you placed it in the code. In most cases, you'll want to place it with more precision using CSS positioning tools. What you need to do is create a definition for the element with the *stage* id. As explained on page 144, that code needs to be in the header for the page or in an external .css file that's linked to the page.

Here's an example:

```
#stage {

    margin: 4px;
    float: right;
    overflow: hidden;
    position: absolute;
    top: 200px;
}
```

CSS provides a number of properties that can be used to position elements on a web page. This definition gives the stage element a margin of 4 pixels. The float command positions the banner on the right side of the page. Overflow is set to "hidden" so elements outside of the stage don't appear. The position property is set to "absolute" which works well if other elements on the page are positioned in a similar manner. You can also use relative positioning if other page elements are "liquid." The last line in the definition moves the banner ad down 200 pixels from the top of the page.

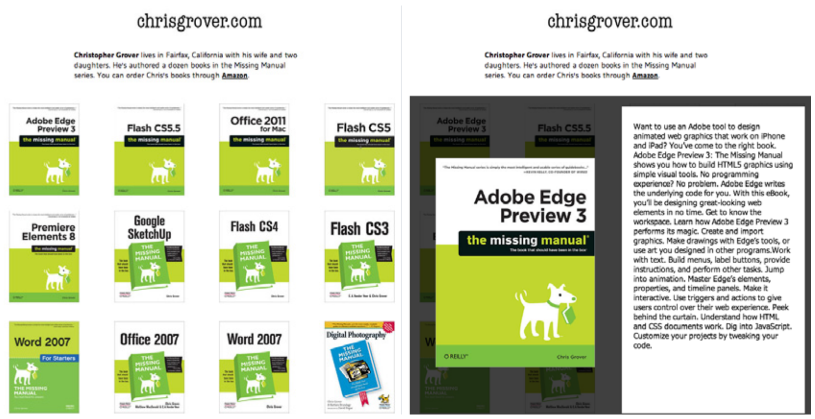
NOTE For more details on CSS and positioning elements with CSS, see David Sawyer McFarland's *CSS: The Missing Manual* (O'Reilly).

Placing Two Edge Compositions on the Same Page

Once you get used to all the features that Edge provides, you'll want to use it for all sorts of projects big and small. Sooner or later, you'll want to place two or more Edge compositions on a single web page. For example, suppose you have a page with an existing Edge composition that displays several book covers. When a web visitor clicks a book, with a little animation magic, it presents a large image of the book cover along with a sales pitch, as shown in *Figure 7-5*. To increase revenue, you decide to add a banner ad for the ever-popular Stutz Bearcat; you previously created this banner ad as a standalone project (page 148). Putting two Edge compositions in one HTML document that's best done outside of Edge in your HTML editor. So, roll up your HTML coder sleeves and get ready for some hand editing. To get started on this project, go to the Missing CD page (*www.missingmanuals.com/cds/edgepv5mm*) and find the *07-4_Two_Compositions* file. It provides the original web page with the books animation. You'll also want to get the *07-3_Banner_Ad* project, which is the second Edge composition that you'll add to the two compositions page.

Here are the chores you need to tackle when placing two Edge compositions on one page:

- In the head of your HTML document, add the preloader code for both compositions.
- In the body of your HTML document, add the <div> code that identifies the two compositions.
- Edit the <div> code so that each composition has a unique ID.
- Use your standard HTML or CSS tricks to position the compositions where you want them on the web page.



chrisgrover.com

Christopher Grover lives in Fairfax, California with his wife and two daughters. He's authored a dozen books in the Missing Manual series. You can order Chris's books through Amazon.

Want to use an Adobe tool to design animated web graphics that work on iPhone and iPad? You've come to the right book. Adobe Edge Preview 3: The Missing Manual shows you how to build HTML5 graphics using simple visual tools. No programming experience? No problem. Adobe Edge writes the underlying code for you. With this eBook, you'll be designing great-looking web elements in no time. Get to know the workspace. Learn how Adobe Edge Preview 3 performs its magic. Create and import graphics. Make drawings with Edge's tools, or use art you designed in other programs.Work with text. Build menus, label buttons, provide instructions, and perform other tasks. Jump into animation. Master Edge's elements, properties, and timeline panels. Make it interactive. Use triggers and actions to give users control over their web experience. Peek behind the curtain. Understand how HTML and CSS documents work. Dig into JavaScript. Customize your projects by tweaking your code.

FIGURE 7-5

Here's a page with an Edge composition. Left: Click on one of the books. Right: With a little animated flourish, the page shows the cover and displays some details about the book. If you want to add another Edge composition such as a banner ad to the top of the page, you have to do some HTML hand-coding.

It's easiest if you start with a web page that already has one composition in place and working properly. That's exactly what you have in *07-4_Two_Compositions*. Open the HTML file in a browser, and you'll see it in action. The results should look like *Figure 7-5*. Here's the step-by-step process for adding the banner ad from page 148 to the top of this existing web page:

1. Examine the HTML and JavaScript files used to display the books page (*07-4_Two_Compositions*).

   The web page has the filename *07-4_Two_Compositions.html*. You see evidence of the existing Edge composition in *books.html*, *books_edge.js* and other JavaScript files that make the project work. In addition, there's an *edge_includes* folder and an *images* folder. If you look in the *images* folder you see a number of large and small JPGs for the book covers.

2. Add the files for the Stutz Bearcat banner ad (page 152) to the folder holding the books project.

   Copy all the files in the *07-3_Banner_Ad* folder to the folder with *07-4_Two_Compositions.html*. Copy all the images in the banner ad images folder to the book project *images* folder. Be careful; you don't want to delete or write over any of the existing images. You're going to need the images for both projects in this one folder. In this case, all the images have unique names. You don't need to copy the files in the *edge_includes* folder; they should be identical, as long as both projects were created with the same version of Edge.

3. Open *07-4_Two_Compositions.html* in your HTML editor. Then find the place in the <head> where the preloader for books is identified.

   It looks like this.

```
<!--Adobe Edge Runtime-->

    <script type="text/javascript" charset="utf-8"
    src="books_edgePreload.js"></script>

<!--Adobe Edge Runtime End-->
```

4. Add a line in between the comments to identify the preloader for your second Edge composition.

   It's easiest to cut and paste lines of code; that way, you're less likely to make a mistake. In this case, you can open the file for the banner (*07-3_Banner_Ad.html*) to copy the line. Then, paste the result into books.html. When you're done it should look like this:

```
<!--Adobe Edge Runtime-->

    <script type="text/javascript" charset="utf-8"
    src="books_edgePreload.js"></script>

    <script type="text/javascript" charset="utf-8"
    src="07-3_Banner_Ad_edgePreload.js"></script>

<!--Adobe Edge Runtime End-->
```

5. In the body portion of the banner ad HTML code, find and copy the line that provides an ID for the <div> tag and paste that into the books.html code.

   You want your banner add to be at the top of the page, so you can paste it immediately underneath the <body> tag, above the rest of the HTML code. For example:

```
<body style="margin:0;padding:0;">

<div id="stage" class="EDGE-225049589"></div>
```

There's still one issue that needs to be resolved. You may remember that an ID attribute like the one shown here must be unique. For example, there can be only one "stage" ID on this page. The solution is to change the names of the two stage IDs to stageOne and stageTwo or something more meaningful to your project.

NOTE There's a number in the class attribute for the <div> that looks like: *class="Edge-225049589"*. This number changes every time you save your project and is used by Edge for its own nefarious, identification purposes. So don't worry if your numbers are different from the ones in this book.

6. Give each composition a unique ID.

   Change the ID to the banner ad:

   ```
   <div id="stage" class="EDGE-225049589"></div>
   ```

   to

   ```
   <div id="bannerStutz" class="EDGE-225049589"></div>
   ```

   Then, change the ID for the book composition:

   ```
   <div id="stage" class="EDGE-6141218">

   </div>
   ```

   to

   ```
   <div id="galleryBooks" class="EDGE-6141218">

   </div>
   ```

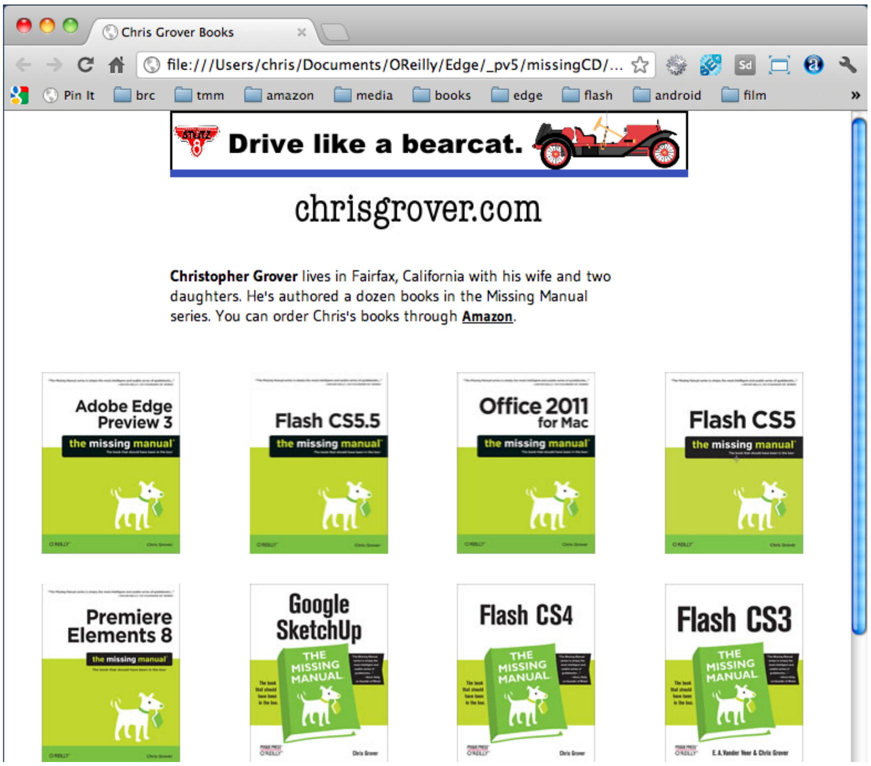
7. Open the *two_comps.css* file in a text or HTML editor. Add a definition for #bannerStutz that creates a 150px left margin.

   The file *two_comps.css* is referenced as and external CSS style sheet in *07-4_Two_Comps.html*. It provides rules for the H1 headers and body text. You can apply CSS rules to position and format the <div> elements that hold your Edge compositions. In this case, adding a 150px left margin positions the banner ad in relation to the other page elements. Here's code for the definition:

   ```
   #bannerStutz {

       margin-left: 150px;

   }
   ```

   The pound sign (#) signals that bannerStutz is an ID. The margin-left is the CSS attribute and 150px is the value.

You can see the finished project with both compositions in place on a web page that includes other non-Edge content in *07-5_Two_Compositions_done*.


Chris Grover Books
file:///Users/chris/Documents/OReilly/Edge/_pv5/missingCD/...
Pin It
brc
tmm
amazon
media
books
edge
flash
android
film
Drive like a bearcat.
chrisgrover.com

Christopher Grover lives in Fairfax, California with his wife and two daughters. He's authored a dozen books in the Missing Manual series. You can order Chris's books through **Amazon**.

Adobe Edge Preview 3 the missing manual
Flash CS5.5 the missing manual
Office 2011 for Mac the missing manual
Flash CS5 the missing manual
Premiere Elements 8 the missing manual
Google SketchUp The Missing Manual
Flash CS4 The Missing Manual
Flash CS3 The Missing Manual

FIGURE 7-6

Here two Edge compositions inhabit one web page. Each composition has a unique ID. The book animation was originally part of the page, which had text and other elements that were not part of the Edge composition. The Stutz Bearcat banner ad was included later.

CHAPTER

8

Controlling Your Animations with JavaScript and jQuery

If you worked on some of the examples in Chapter 5, you've already dabbled in JavaScript and jQuery. When Edge creates triggers and actions, it produces JavaScript/jQuery code. Modern browsers all understand this code, and it works universally, unless someone has explicitly turned JavaScript off. Like the CSS code mentioned in Chapter 7, JavaScript can be interspersed within the HTML code for a web page, or a separate JavaScript (.js) file can be can be linked to the page.

This chapter isn't meant to be a complete study of JavaScript and jQuery, but it is meant to help you get started. You'll learn how to read the code that Edge produces, and you'll learn how you can tweak that code to customize your Edge compositions. Throughout, you'll find tips and techniques that help solve common issues when you're working in Edge. This chapter focuses mostly on JavaScript and jQuery theory. Chapter 9 explains how to put this theory into action.

A Very Brief History of JavaScript and jQuery

Once upon a time (in the 1990s), there was a company called Netscape, which delivered one of the first widely used web browsers, Netscape Navigator. Soon, the company was in a death battle with another company called Microsoft, which put forth a competing browser called Internet Explorer. In an effort to keep a competitive edge over their nemesis, some Netscape wizards developed a scripting language that could be used to add automation and interactive features to web pages. The language had different names at different times, like Mocha and LiveScript, but the name that stuck was JavaScript. JavaScript was welcomed and widely used by the web-making masses. It proved to be so popular that Microsoft developed its own version, JScript...but that's another tale. At times, JavaScript suffered from all-too-familiar tech ailments: competing standards and implementation conflicts among browsers. (Translation: Microsoft's Internet Explorer behaves differently from every other browser.) As time went by, many of these issues were smoothed over. Support for JavaScript became more consistent among browsers. JavaScript libraries were developed, which made it easier for web designers to focus on building pages rather than dealing with browser inconsistencies. jQuery is one of the most popular libraries that serve this function. (Pardon the pun.) jQuery helped solve many other issues for JavaScript coders, truly living up to its motto: Write less, do more. With the advent of HTML5, many new and powerful tools were available for web building, and when combined, they were very powerful indeed. Web builders were happy, and the web page audience enjoyed visually entertaining and interactive experiences. Sadly, Netscape didn't survive, but JavaScript lives on.

JavaScript Versus ActionScript and Other Languages

If you're familiar with Flash and now you're adding Edge to your web-building toolbox, you're not alone. You'll find that JavaScript has a lot in common with ActionScript, Flash's programming language. Technically, they're both *scripting* languages, meaning that they're programming languages that run inside other environments—like web pages. On top of that, JavaScript and ActionScript both share the same programming language specification, ECMA-262.

NOTE Since you're just dying to know, ECMA stands for European Computer Manufacturers Association, the standards group that established the spec.

Initially, programmers used both JavaScript and ActionScript in snippets to perform quick and easy chores. These snippets are similar to the triggers and actions in Edge. For example, in JavaScript, you'd write something like the following:

```
on (press) {
 startDrag(this);
}
```

Often, you'll find JavaScript interspersed throughout the HTML code that describes web pages. From a technical point of view, JavaScript and ActionScript are considered high-level languages, because they're closer to human language than the 1s and 0s of machine language.

Sleuthing Through the JavaScript Edge Creates

If you want to create a simple animation that runs from start to finish, you don't have to dig into JavaScript/jQuery code. However, if you want to add interactivity to your project or perform other magical feats, you'll want to add triggers and actions as explained in Chapter 5. If you want to modify those triggers and actions, you need to learn something about the way JavaScript and jQuery work. At first, programming and animation may feel like a curious match, since artists and programmers often seem to be such different people. But when you think about it, programming and drawing are both creative activities. Just like an artist, a programmer needs imagination and vision. And animation is a very programmatic visual art, complete with reusable chunks of action that branch off into separate scenes. On large creative teams, you'll find some people responsible for artwork and others responsible for code. However, there are plenty of small shops where the same person handles both duties.

One of the best ways to learn how JavaScript works is to read code. That's easy enough to do, and it's exactly where this chapter starts. You can begin by examining the code that Edge creates when you save a project. Then add an element or make a change, and examine the new code that Edge creates. Bit by bit, you can learn how certain chunks of code affect your project.

Follow these steps to see how this technique works:

1. **Create a new Edge project and immediately save it with the name *Empty*.**

   Edge creates HTML and JavaScript files.

2. **Go to File→Preview In Browser.**

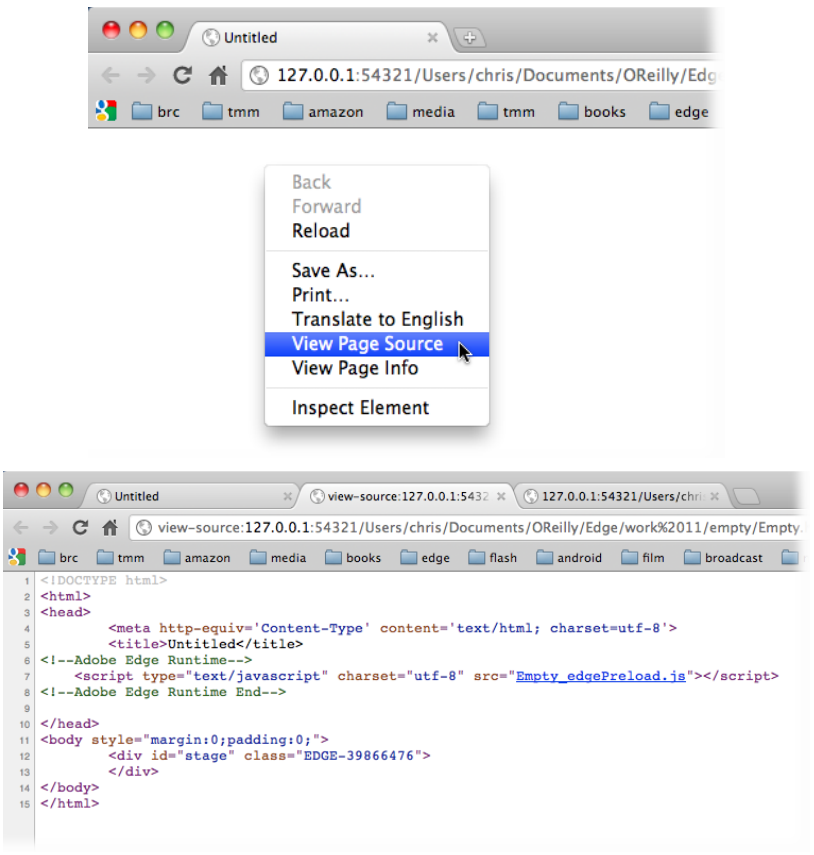
   Your project opens in your web browser. Surprise—you see an empty page.

3. **Right-click (Control-click) the web page in your browser and choose View Source (or View Page Source).**

   Most browsers show the HTML code for a web page in a separate tab or window.

When you examine the source code for the empty web page, there's nothing much new that wasn't discussed in previous chapters. There's a <script> reference to the JavaScript preloader, *Empty_edgePreload.js*. The preloader is responsible for linking your project to all the resources it needs. Many of those are JavaScript libraries. For details, see the box on page 159. You won't be messing with the files in your *edge_includes* folder. Those are standard libraries, like jQuery. You want to examine the two JavaScript files that are specific to your project. Conveniently, they're in the same folder with the .html file for the page and the first part of the filename matches your project. So, as you see in the bottom of *Figure 8-1*, if your project name is "Empty," then the preloader filename is *Empty_edgePreloader.js*. If you inspect the preloader code at the bottom, you see the loadResources function that links to other JavaScript files. You can examine JavaScript files in some web browsers or in a text editor like Notepad (Windows) or TextEdit (Mac).

```
loadResources([
    { load: "edge_includes/jquery-1.7.1.min.js"},
    { load: "edge_includes/jquery.easing.1.3.js"},
    { load: "edge_includes/edge.0.1.5.min.js"},
        {test: !hasJSON, yep:"edge_includes/json2_min.js"},
          { load: "Empty_edge.js"},
          { load: "Empty_edgeActions.js"}], doDelayLoad);
```



```
<!DOCTYPE html>
<html>
<head>
        <meta http-equiv='Content-Type' content='text/html; charset=utf-8'>
        <title>Untitled</title>
<!--Adobe Edge Runtime-->
    <script type="text/javascript" charset="utf-8" src="Empty_edgePreload.js"></script>
<!--Adobe Edge Runtime End-->

</head>
<body style="margin:0;padding:0;">
        <div id="stage" class="EDGE-39866476">
        </div>
</body>
</html>
```

FIGURE 8-1

Top: All popular web browsers let you view the HTML source code for a web page. The command in Chrome, shown here, is Right-click→View Page Source.

Bottom: The source shown here lists the preload script that Edge adds to the web page. In turn, the preload script links several JavaScript libraries to the page.

TIP Most browsers give you a way to view a web page's HTML source code. The command is usually View Source or View Page Source. If you're using Google Chrome for a browser, then after you choose View Page Source, you can click on the JavaScript filenames in the displayed code to view the JavaScript files in a new tab, as shown in *Figure 8-2*. This is a quick, handy way to study code, but you can't edit the files. Both Google Chrome and Mozilla Firefox have great developer tools that help you examine web pages and the files that make them work.

POWER USERS' CLINIC

Edge Project Files

As explained on page 11, when you save a project, Edge creates several files even if you don't do anything. What exactly are those files, and what do they do? Here's a quick rundown. Suppose you create and save a project named *Empty*. In the HTML file named *Empty.html*, you'll find a <script> tag that loads *Empty_edgePreLoad.js*. In turn, that preloader links to these files, though the version numbers may be different:

jquery-1.7.1.min.js. The official jQuery library. It includes all the routines that make it easy for you to write code that works with the whole spectrum of browsers that are out there. In addition, jQuery makes it easier to identify the elements on a web page. See the details on selectors, page 139.

jquery.easing.1.3.js. A plugin to the jQuery library that handles all those cool easing effects that you can use with transitions. This is a standard library, so if you're a JavaScript/jQuery wiz, you can use it outside Edge.

edge.0.1.5.min.js. The Edge JavaScript library. When you create animations and other effects, the JavaScript code created refers to methods and functions in this library.

edge.symbol.0.1.3.min.js. Symbols are a core feature in Edge projects, and the library to handle symbols is separate from the basic Edge library.

json2_min.js. A specialized JavaScript library that most often serves as a data interchange format.

If you want your web pages to load and display quickly in browsers, you want to minimize the amount of data that's loaded before the page appears. The JavaScript files with *min* in their names have been "minified," or made as small as possible. Upside: They load fast. Downside: They can't easily be read by humans. If you want to examine the readable jQuery source code, head over to *http://jquery.com*.

In general, you don't need to worry about the contents of these files. In fact, you're explicitly warned not to make any changes to them. They are safely tucked away in the *edge_includes* folder.

The other files that Edge creates are in the same folder as the HTML web page document. You can and should examine these files. As explained in this chapter, you can even tweak them a bit to make changes to your project. If you create the Empty project, you'll find:

Empty_edge.js. This is the file that defines the stage, Timeline, and elements in your project. It defines the base state or starting point for each element. When you create a *transition*, the properties for your element transition *from* one state *to* another state over a period of time (*duration*).

Empty_edgeActions.js. When you add triggers and actions to your Edge composition, the details are stored in this file. If you want to tweak or customize those actions, you can do it in the Edge code panels, or you could make changes here.

Empty_edgePreload.js. The preloader file that's responsible for loading the resources and scripts Edge needs to perform its magic.

The first time you open the *Empty_edge.js* file, it may be a little confusing. There are lots of strange terms and characters, like { }, ( ), and $, sitting on indented lines. It starts off like this:

```
/**
 * Adobe Edge: symbol definitions
 */
(function($, Edge, compId){
var fonts = {
};
var symbols = {
"stage": {
   version: "0.1.4",
   baseState: "Base State",
   initialState: "Base State",
   gpuAccelerate: true,
   content: {
      dom: [
],
      symbolInstances: [

      ]
   },
   states: {
      "Base State": {
         "${_stage}": [
            ["color", "background-color", 'rgba(200,200,200,1)'],
            ["style", "height", '400px'],
            ["style", "width", '468px']
         ]
      }
   },
        ...the .js file continues but is not shown here...
```

If you're new to this, try the drowning sailor technique: Grab hold of anything that looks recognizable. Comments are always good. (For a refresher on comments, see the box on page 98.) There's a block comment at the top that says, "Adobe Edge: symbol definitions." That's pretty clear. The code that follows must define the symbols in the project.

Down around line 8 you see:

```
"stage": {
```

It seems likely that this is a reference to Edge's stage. Lines that follow appear to define the stage. There's a reference to *baseState* and *"Base State,"* which sound suspiciously as though this code may set the starting values for specific properties for the stage. Sure enough, down around line 18, there are more Base State details, including height, width, and background-color. Edge master that you are, you remember that these are properties you can set in the stage's Properties panel. In fact, at this point, if you're daring, you can edit these properties from inside this JavaScript file. For example, if you change the line:

```
["style", "height", '400px'],
```

to:

```
["style", "height", '800px'],
```

and then save the *Empty_edge.js* file, then the next time you open *Empty.html* in a browser the stage will be 800 pixels tall.

TIP If you edit your edge project's JavaScript file while its still open in Edge, you tend to confuse Edge. You'll see a message noting that the file has been changed and you'll be asked if you want to reload the file. If you say Yes to reload the file, you load the file that was saved outside of Edge and you lose any change you made in Edge. If choose No, you won't see any changes in Edge. However, if you save the file, you'll overwrite any changes you made outside of Edge.

The next chunk of code appears to set values for the base state of the Timeline. It looks like this:

```
timelines: {
      "Default Timeline": {
         fromState: "Base State",
         toState: "",
         duration: 0,
         autoPlay: true,
         labels: {

         },
         timeline: [
]
      }
   }
```

The *duration*, *autoPlay*, and *labels* properties appear to be represented, and their values look about right for an empty Edge project. They're empty or set to zero.

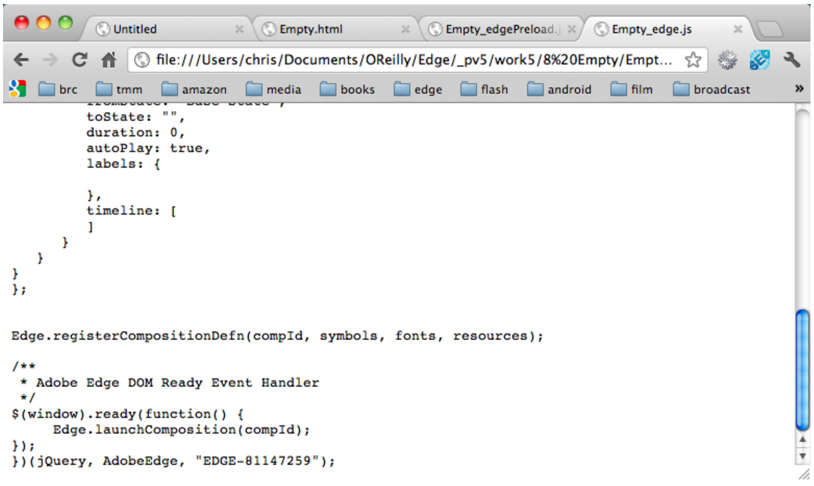
As shown in *Figure 8-2*, the code near the bottom of the file, beginning with:

```
Edge.registerCompositionDefn(compId, symbols, fonts, resources);
```

...and through to the end, is used to set up the composition (Edge project). It checks to see that everything is loaded and that the browser window is ready. When it's ready, it runs the launchComposition method.

```
/**

 * Adobe Edge DOM Ready Event Handler

 */

$(window).ready(function() {

     Edge.launchComposition(compId);

});

})(jQuery, AdobeEdge, "EDGE-81147259");
```

The more code-sleuthing becomes a habit, the faster you'll learn JavaScript and its partner in crime jQuery. If you carefully duplicate and save projects, you can experiment with the code just to see what happens. If you mess up a duplicate file or project—no problem. Just delete it and try again.



```
        toState: "",
        duration: 0,
        autoPlay: true,
        labels: {

        },
        timeline: [
        ]
      }
    }
}
};

Edge.registerCompositionDefn(compId, symbols, fonts, resources);

/**
 * Adobe Edge DOM Ready Event Handler
 */
$(window).ready(function() {
     Edge.launchComposition(compId);
});
})(jQuery, AdobeEdge, "EDGE-81147259");
```

FIGURE 8-2

In Google Chrome, you can open a web page, then view the HTML code. If there are links to external JavaScript files, a click will display the code in a new tab, as shown here. The code at the bottom of this file registers Edge projects and plays the composition when the browser is ready.

Adding a New Element to the Stage

The next step in the sleuthing expedition is to add a new element to the stage and see how that changes the code in *Empty_edge.js*. It's best to start simply, so you don't get lost in a sea of code, so just add a square that's 100 x 100 px and name it Square.

TIP Hold the Shift key down when you drag out a rectangle to create a square. Keep your eye on the Properties panel, and it's not hard to create a 100 px square.

Open the *Empty_edge.js* file in a code editor or plain text editor and you see the new code that defines your square. Under these lines that were already in the project when it was empty:

```
content: {
      dom: [
```

you see:

```
{

            id:'Square',

            type:'rect',

            rect:[147,89,100,100],

            fill:["rgba(147,207,248,1.00)"],

            stroke:[0,"rgba(0,0,0,1)","none"]

         }],
```

The first few lines are pretty clear. There's an element with the ID of Square. That's the name you gave to the rectangle. The next line confirms that the element is the "rect" type. The other lines describe the rectangle properties you'd find in the Properties panel (*Figure 8-3*). The next three lines describe the appearance of the rectangle. For example:

```
rect:[147,89,100,100],
```

The first two numbers set the position of the rectangle (X=147, Y=89). The next two numbers describe the width and height. Following that, the fill and stroke properties are set in *rgba* terms: red, green, blue, and Alpha.

```
fill:['rgba(192,192,192,1)'],
```

In this case, the fill is light blue, because of the varied red, green, and blue values. With Alpha set to 1, the square is 100 percent opaque. Want to change your square to be red? Edit that line to read:

```
fill:['rgba(255,0,0,1)'],
```

Want to make the square semi-transparent? Change that last number (a) to a decimal, like this:

```
fill:['rgba(255,0,0,.5)'],
```

Now, there may not be a great value in being able to edit values like these on a new Edge project; you can easily make those changes in Edge when you begin a project. But there is a value in experimenting to see how changes like these affect elements in your project. The things you learn at this stage can be applied to other chunks of code you create using triggers and actions. From time to time, you may get JavaScript code from members of your project team or other sources. The more fluent you are in JavaScript/jQuery, the better you'll be at solving problems and building masterpieces.

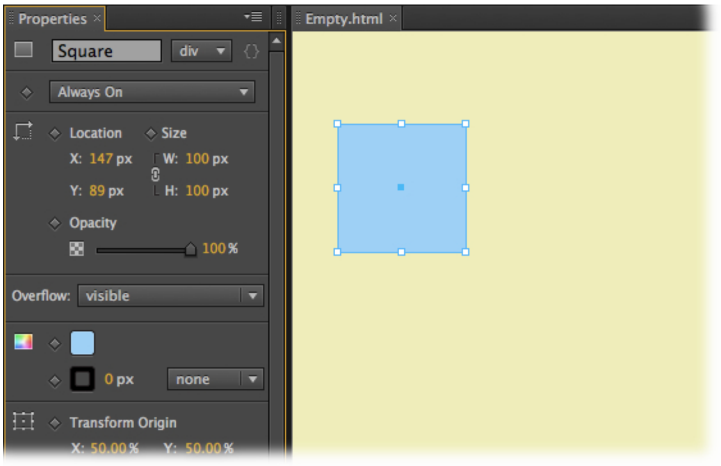


FIGURE 8-3

You can find the same properties that define a square's position, size, and color in the JavaScript code that Edge creates. Tweak the code, and you change the way your square looks in Edge and on the web page.

JavaScript and jQuery Basics

JavaScript shares many programming concepts with other popular languages. If you have any programming experience, from C to Java to ActionScript, you certainly have a head start with JavaScript. This section covers some of the JavaScript and jQuery basics that are helpful to know when you're working with Edge. It's certainly not a complete reference or a tutorial on the subjects. If you're looking for a complete study of JavaScript and jQuery, that'll take an entire book. The name of that book happens to be *JavaScript and jQuery: The Missing Manual* by David Sawyer McFarland (O'Reilly). You can find online information about JavaScript at *www.w3schools.com/js* or *https://developer.mozilla.org/en/JavaScript*.

When it comes to types of data, JavaScript provides the usual suspects:

- **Numbers** can be whole numbers like *756* or decimals like *7.56*. Numbers are expressed without any particular punctuation. For example, you might have a statement like:

  ```
  var myNumber = 12 + 5;
  ```

  The 12 and 5 are added together, and the result is stored in the variable myNumber.

- **Strings** are groups of characters, like your name or the words "Moby-Dick." Strings must appear inside quotes, either single or double. Statements with valid strings might look like this:

  ```
  var author = "Herman Melville";
  var firstSentence = 'Call me Ishmael.';
  ```

  You can combine two strings using the + operator. For example:

  ```
  var author = "Herman " + "Melville";
  ```

  Stores a single string "Herman Melville" in the *author* variable.

- **Booleans,** as usual, have two values: *true* or *false*. They're often used to determine if a certain condition exists. Here's an *if* conditional statement that checks to see if the sun has come up:

  ```
  sunRise = true;
  if (sunRise == true) {
   newDay();
  };
  ```

  The first line assigns the value "true" to the variable sunRise. If it's true that the sun has come up, then the *newDay()* function is performed. The details of *newDay()* are defined elsewhere. It's important to remember that the operator to compare equal values is ==, not a single =.

- **Arrays** provide a convenient way to group several items together. For example, an array of colors might look like this:

  ```
  var colors = ['red','blue','green','alpha'];
  ```

  In that case, you could retrieve values from the array with a statement like this:

  ```
  myFavorite = colors[1];
  ```

  The numbering for arrays always begin with 0, so in this example *colors[1]* references *blue*.

  A single array can hold different types of data. So for example, an array for an employee could have strings that store first and last names and numbers that store age and number of years employed. There are a number of ways to add items, remove items, and change items in an array. Details are available online at *www.w3schools.com/js* or *https://developer.mozilla.org/en/JavaScript*.

- **Functions** consist of one or more statements that can be executed. You may be surprised to see functions listed among the *data types*, but in JavaScript, functions are considered a data type. As a result, functions have some interesting capabilities. For example, they can be stored in variables or arrays. A function might be declared like this:

  ```
  function fullName(first, last) {
  return first + ' ' + last;
  }
  ```

  The word "function" explains that the code that follows is a function. The name of the function is *fullName,* and it takes two arguments: *first* and *last*. These arguments need to be provided when the function is called. One or more statements for the function appear in between curly braces { }. This function joins three strings to create one string. The arguments *first* and *last* are strings, presumably the first and last names. In between, a *string literal* is used to insert a space character. Code to call the function might look like this:

  ```
  name = fullName(firstName, lastName);
  ```

  In this case, the variables *firstName* and *lastName* are presumed to be strings that have already been defined to represent someone's name. So if previously the variable firstName was given the value *Umberto* and lastName was given the value *Eco*, then this statement would assign the string "Umberto Eco" to the variable *name*.

Here are some other features you should keep in mind when you work in JavaScript:

- **JavaScript is loosely typed.** Unlike many other languages, variables do not have a specific data type when they are created. This means a variable could be assigned a number value and then later a string value.

- **JavaScript is case sensitive.** Capitalization matters in JavaScript. That means "javascript," "JavaScript," and "javaScript" could be used to name three separate variables.

- **Variable names must be specific characters.** Variables must begin with a letter, $, or _. They cannot begin with a number. The rest of the characters in a variable must be letters, numbers, $, or _. Variables can't contain any other characters or punctuation.

- **JavaScript has reserved words.** Like most programming languages, JavaScript has a number of words that have special meanings. These are usually called *reserved words* or *keywords*. You don't want to confuse the process by using these words for variables or in other places. The following are examples of reserved words:

  - *var*—used to create variables.
  - *if*—used to begin if...else conditional statements.
  - *while*—used to begin while conditional statements.

  - *new*—creates a new instance of an object.
  - *function*—used to define a function.

  You can find the complete list of JavaScript reserved words at *https://developer.mozilla.org/en/JavaScript/Reference/Reserved_Words.*

- **JavaScript uses semicolons (;) to end statements.** To properly interpret JavaScript code, browsers need to know where one statement ends and the next begins. JavaScript uses semicolons to separate statements. If there's only one statement on a line, many browsers will let you get away with not putting a semicolon at the end, but it's considered a best practice to always put a semicolon at the end of a statement.
- **JavaScript ignores whitespace.** In general, JavaScript ignores spaces, tabs, and new lines. That leaves you free to add whitespace to your code to make it more readable. Naturally, if you put whitespace in the middle of something like a number, that would change *736* into two numbers, *7 36*.

Operators in JavaScript

Programs and scripts perform their magic through a sequence of operations where variables are assigned values, calculations are performed, and comparisons are made. These operations require operators like + for addition and / for division. The operators used in JavaScript are similar to those used in other programs and scripting languages.

The assignment operator is =. You use this to assign a value to a variable, as in:

```
age = 23;
name = "James Joyce";
```

Basic math operations use these standard operators:

OPERATION	OPERATOR	EXAMPLE
Add numbers	+	45 + 21
Subtract numbers	-	45 - 21
Multiply numbers	*	45 * 21
Divide numbers	/	21/7

You can use these operators with literals—that is, actual numbers—or you can use them with variables that have number values assigned to them. Sometimes you'll end up with a string that is a numeral, as in:

```
age = "23";
```

The variable "age" has been assigned a string. You can tell by the quote marks. That string happens to be numerals. What if you want to perform a math operation with *age*? JavaScript gives you an easy way to convert that string to its numerical value.

Just put a + sign in front, like so:

```
doubleAge = +age * 2;
```

Then you can calculate to your heart's content.

A special set of operators is used to compare values. The trickiest one is the equal-to operator. It's easy to forget you want two = signs, as in ==.

Here are the comparison operators. The result of each example is true:

OPERATION	OPERATOR	EXAMPLE
Are two values equal?	==	(4+1)==5
Are two values unequal?	!=	(4+1)!=3
Is the left value greater than right value?	>	24 > 12
Is the left value greater than or equal to the right value?	>=	24 >= 12 24 >= 12 + 12
Is the left value less than the right value?	<	12 < 24
Is the left value less than or equal to the right value?	<=	12 <= 24 12 <= 6 + 6

Conditional Statements in JavaScript

Comparison operators are often used in conditional statements. For example, as a programmer for the Department of Motor Vehicles you might say, "If this applicant's age is less than 16, then deny driver's license." Written in JavaScript, the statement might look like this:

```
If (applicantAge < 16) {
    denyLicense();
    alert('License is denied, due to youthful recklessness!');
}
```

Understanding the Document Object Model (DOM)

JavaScript gives you a way to modify and rewrite HTML documents. JavaScript does this by accessing elements in the document and then making changes. CSS formatting works in a similar way when it applies formatting to individual elements in a web page. In effect, it amounts to rewriting the HTML that defines the page. The document object model, affectionately known as the DOM, is the skeleton of any web page, as visualized in *Figure 8-4*. It's a conceptual definition of a web page that describes the elements.

Before you can make changes to an element in an HTML document, you need to identify or *select* that element. There are three common ways to identify the elements in a web page: by tag, by ID, or by class.

- **Tags** are HTML's basic method for identifying things like headings <h1> or paragraphs <p>. Tags always use angle brackets: < >.
- **IDs** are used to identify one unique item on a page. An ID can be assigned to an element using an attribute within a tag. For example, a picture could be given an ID of mainPhoto within the <img> tag:

```
<img id="mainPhoto" src="images/bearcat.jpg">
```

- **Classes** are used to identify similar elements on a page. To identify a photo as part of the "gallery" class, you'd write:

```
 <img class="gallery" src="images/bearcat.jpg">
```

JavaScript has methods for getting an element by ID and for getting an element by tag. It also provides ways to move around the DOM to select different elements. This technique is called traversing, and if you're not used to using it, it's likely to cause headaches. However, there's good news and more good news: jQuery provides ways to select elements that are much easier to use. And, for the most part, Edge uses jQuery methods to select elements.

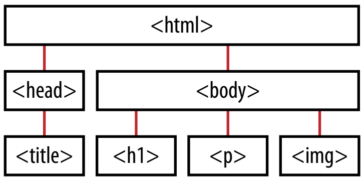


FIGURE 8-4

A web page is the sum total of its elements. There's a certain hierarchy, as some elements reside inside others. Here, for example, all the elements are inside html, and the paragraph or <p> tag is inside <body>. There are many elements defined by the DOM. They aren't all shown here.

Natural Selection the jQuery Way

In Edge, when you name an element, something is going on behind the scenes. Edge is giving that element an ID. So if you name a photo *mainPhoto* in Edge, it's just as if you'd written *id="mainPhoto"* in the <img> tag. Remember how CSS references IDs by placing a # (pound sign) in front of the name? (For a reminder, see page 141.) jQuery uses the same system. If you want to reference your *mainPhoto* using jQuery, you'd write:

```
$('#mainPhoto')
```

Remember how CSS references classes by placing a period in front of the name? jQuery uses that system to identify classes, too. If you want to select all the elements in the gallery class, it looks like this:

```
$('.gallery')
```

Finally, if you want to select all the elements with <img> tags, you don't need any special character at the front. That would look like this:

```
$(img)
```

"this" and "sym" Are Special Words

In JavaScript, the word "this" has a special meaning. A great deal of your effort in writing and editing JavaScript and jQuery code involves identifying exactly the right element so that you can do something to it. You may want to move it to a new position. You may want to change its color. You may want to replace all the text that it displays. The reserved word "this" is often used to identify an element that has been clicked or the object from which a function (method) is called. However, "this" is more flexible in JavaScript than it is in other languages, so its use may surprise you. In Edge, "this" usually refers to the symbol and the *lookupSelector()* function may be used to identify specific element—in other words, children of that symbol. For example, this statement will identify an element named "text1" and then hide it:

```
$(this.lookupSelector("text1")).hide();
```

Early versions of Edge used this code in the Actions panel, and it still works. However, later versions of Edge, including Preview 5, have simplified the code for the Hide action to:

```
sym.$("Text1").hide();
```

In this case, "sym" stands for "symbol." The dollar sign is the jQuery selector that looks up the element by name (ID). The .hide() function remains the same. You can add this statement using the Actions panel and then edit it for your own purposes. Change text1 to the name of the element you want to identify. Then, if needed, change the function hide() to one that you want to use. One great advantage of this "build-it-yourself snippet" is that you'll cut down on typos in your code.

CHAPTER

9

Helpful JavaScript Tricks

Here's the payoff for learning the JavaScript/jQuery basics covered in Chapter 8. You can apply those skills to your Edge projects. One of the easiest ways to start is to use Edge's triggers and actions to write code, and then apply your own modifications to customize that code. That's exactly how this chapter begins. Then it presents yet another version of the photo gallery, using a nifty image source swapping technique. Along the way you learn how to change an element's dimensions, position, and background colors the JavaScript/jQuery way. This chapter explains how you can simplify your code by assigning symbols and elements to variables. Last, but not least, you'll see how to control symbols' independent timelines.

More Showing and Hiding Tricks

Chapter 5 showed several examples using the Show and Hide actions. Now it's time to revisit that subject and see how you can bend those simple tricks to your own iron will. Consider a simple Edge composition like the one in *Figure 9-1*. You can find *08-1_HideAndSeek* with the exercises for this chapter on the Missing CD at *www.missingmanuals.com/cds/edgepv5mm*.

The idea is to make the text boxes with the words Show, Hide, and Toggle behave like buttons. Click on the word and the car picture disappears or reappears. For example, as covered in Chapter 5, this is the quick way to make Hide work:

1. In Edge, right-click (Control-click) the word "Hide" and choose Open Actions for "tbHide" from the shortcut menu.
2. Choose *click* from the triggers menu.

3. In the Actions panel, click the Hide Element button on the left.

NOTE In this example, the letters tb are added to the front of each element that is a text box. As your projects grow, you may want to use similar techniques to further distinguish the many elements inside. Two or three letter prefixes or suffixes work well.

That inserts the generic hide code, *Figure 9-2*, which looks like this:

```
sym.$("Text1").hide();
```

You need to change "Text1" to *car* so that a click on the word hides the image of the car. When you're done, it reads like this:

```
sym.$("car").hide();
```

You go through a similar process when you create a trigger for the text box with the word "Show." Just make sure you use the Show action.

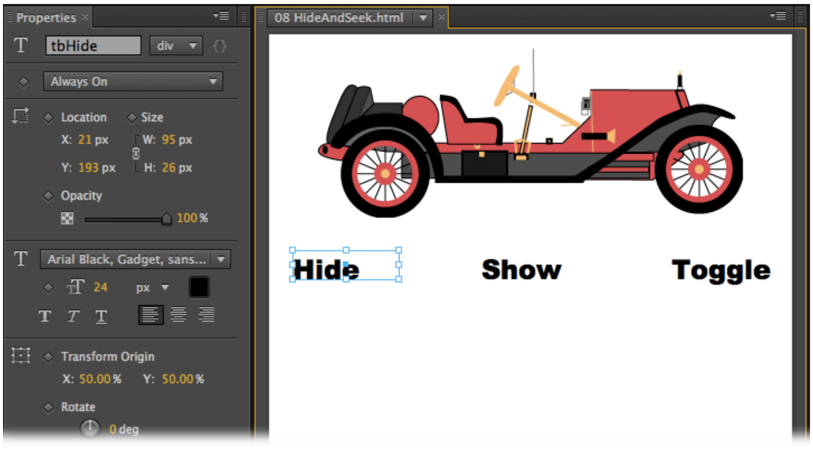


FIGURE 9-1

This Edge composition has an image of a car and three text boxes with the words "Hide," "Show," and "Toggle." Guess what happens next.

NOTE Check back on page 108 if you need to brush up on showing and hiding elements using triggers and actions.

So what about that word "Toggle"? It can be awfully handy to have a single button or command to turn an element on the web page on and off. You could write something with one of those conditional statements like, "If the car is visible, hide the car; else show the car." But there's an even easier way. jQuery has a companion to the show and hide functions called *toggle*. All you need to do is follow the steps above to apply a *click* trigger and a *hide* action to the car element. Then you can edit the statement. You want to change the *hide* action to a *toggle* action. When you're done, the code for Toggle looks like this:

```
sym.$("car").toggle();
```

Save your work and give it a test. If the car is visible, clicking either Hide or Toggle makes it disappear. Likewise, if the car is hidden, clicking Show or Toggle makes it reappear.

Delaying Action for Show, Hide, and Toggle

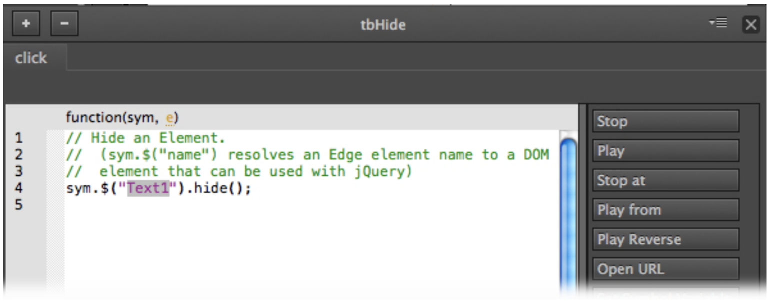
If you read through the beginning of this chapter, you may recognize that *show(), hide(),* and *toggle()* are functions. Somewhere deep in the JavaScript code that's linked to your Edge composition are the routines that make these functions perform. When your browser comes across a function name like *hide()*, it looks up those routines and applies them to the car element. You don't have to know the nitty-gritty coding details that make a function work; it's enough to understand what they do and how to use them. It's like the difference between driving a car and being an auto mechanic.

Here's one more hide, show, and toggle trick. You can add *arguments* (sometimes called parameters) to the *hide(), show(),* and *toggle()* functions that change the way they behave. Open the click actions for the text box "hide." Change the code by placing the number *4000* inside the parentheses after "hide." The statement should look like this:

```
sym.$("car").hide(4000);
```

Do the same for the *show()* function of the text box with the word "Show." Now test the composition. When you click on "Hide" and "Show," the box gradually shrinks from view or grows into view from the upper-left corner. You may have seen this technique used to show and hide menus on a navigation bar. Without any argument, "Toggle" still works instantaneously.

You may recall that JavaScript likes to divide time into milliseconds—that is, thousandths of a second. That's exactly what's happening in your *hide()* and *show()* functions. It takes 4 seconds for the shrinking and growing to take place. If you'd rather use words to specify the timing, try "slow," "normal," and "fast." Don't forget to include the quote marks.



```
function(sym, e)
// Hide an Element.
//  (sym.$("name") resolves an Edge element name to a DOM
//  element that can be used with jQuery)
sym.$("Text1").hide();
```

FIGURE 9-2

When you use Edge's triggers and actions, you often need to modify the code for your needs. For example, here the generic hide code needs to be edited to identify "car" as the element to hide.

Fading In and Out

As the infomercial salesperson likes to say: But wait! There's more! The shrinking and growing from the previous example works well for some chores, like menus. For photos and other images, a nice dissolve animation works well. Sure, you can create this effect on the Timeline and use "Play from" triggers, but why bother if you can do the same thing by swapping functions?

If you want to keep using the same composition, you can change the words in the text boxes to read "Fade Out" and "Fade In" or you can add new text boxes to do the job. You can continue to use *click* as the trigger. The action uses new functions:

- *fadeOut()*: The element dissolves (becomes less opaque) until it disappears.
- *fadeIn()*: The element increases in opacity until it is fully visible.
- *fadeToggle()*: Shows or hides the element by changing its opacity.

Keep in mind, as always, JavaScript and jQuery are case sensitive, so be sure to fadeOut() rather than FadeOut or fadeout. As with the *hide()* and *show()* functions, you can use arguments to control the speed. These fading actions accept both numbers (in milliseconds) and the words "slow," "normal," and "fast."

Slip Sliding Up and Down

Order now and you'll get sliding action, too! You guessed it: You can swap these functions in place of *show()* and *hide()*.

- *slideUp()*: Slides an element up until it disappears.
- *slideDown()*: Slides an element down until it is fully visible.
- *slideToggle()*: Shows or hides the element using the sliding action.

As with the *hide()* and *show()* functions, you can use arguments to control the speed. These sliding actions accept both numbers (in milliseconds) and the words "slow," "normal," and "fast."

If you completed all the steps, you'll have a hide-and-seek sampler that you can view in your browser, *Figure 9-3*. If you haven't followed along, you can find *08-2_HideAndSeek_done* on the Missing CD at *www.missingmanuals.com/cds/edgepv5mm*.

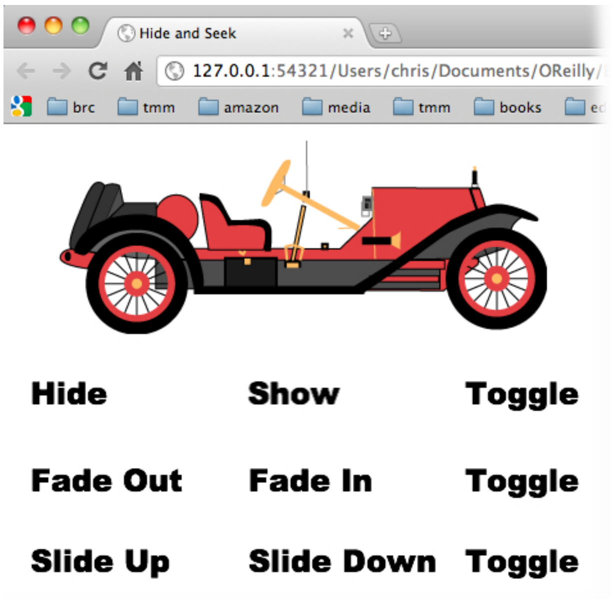


FIGURE 9-3

The text boxes in this example all work like buttons. Click on a word to see how the function affects the image of the car. Examine the code, and you'll see the different delay settings that control the timing of the animated effect.

FREQUENTLY ASKED QUESTION

The Fickle Finger of Click

Why don't I see that pointing finger when my cursor moves over a clickable item?

We're all well-trained. We expect to see that pointing finger when we move our cursor over a button or a hyperlink. Yet when you create clickable elements in Edge, that's not always the case.

There is a JavaScript/jQuery fix to this problem, and it involves only one piece of code. First right-click (Control-click) an element in your project, and then add the *mouseover* or *vmouseover* trigger. Then add a line of code that looks like this:

```
sym.$('tbHide')).css('cursor','pointer');
```

In this example, *tbHide* is the ID for the text box that holds the word "Hide." It is the element with the *mouseover* trigger. With this action in place, the cursor changes to a pointing finger when it's over the text box.

In later versions of Edge, the Adobe wizards made this process even easier. Select an element on the stage, then, in Properties, click the cursor button. A menu flies out and displays a number of different cursor options, as shown in *Figure 9-4*. Click one, and that cursor is automatically displayed when the mouse's cursor is over that element.

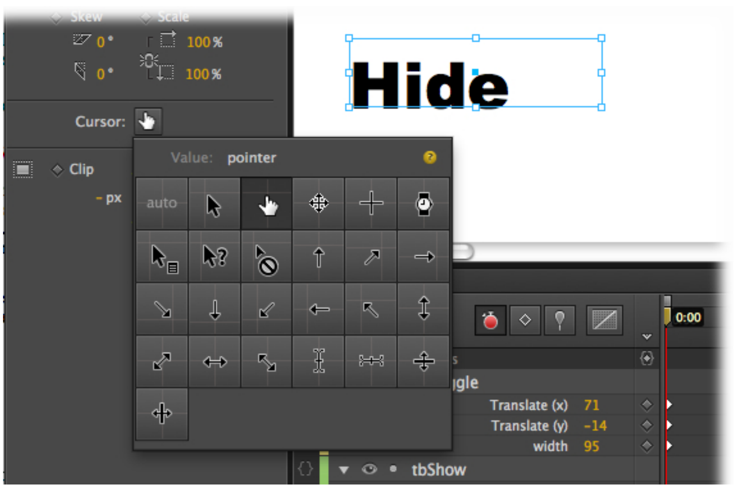


FIGURE 9-4

In Properties, Edge has a nifty feature that makes it easy to specify a cursor style that's used when someone mouses over an element on the stage.

More Visual Effects with *animate()*

As explained in Chapter 7, formatting chores for web pages can be handled by CSS (Cascading Style Sheets). JavaScript and jQuery statements can rewrite the CSS rules and, in the process, change the way elements appear on a web page. The *animate()* function does just that. For example, using *animate()*, you can change the opacity of the car image, or you can change the appearance of the text in a text box.

The *animate()* function works a little differently from the previous ones, because it can use several arguments at once. For example, you can change the opacity of an image and crop the image at the same time. To do that with the car image, you'd write the function like this:

```
sym.$("car").animate(
{
   opacity: .25,
   width: 200
}
);
```

There are a few new things to note in this example. The *animate()* function holds arguments inside parentheses like any self-respecting function. The opening parenthesis is at the end of line 1, and the closing parenthesis is at the beginning of line 6.

Inside those, there are curly braces that hold multiple arguments. In this case, the arguments specify the opacity and width of the image. These arguments match up with standard CSS specs, and they're separated by commas. The function is considered a single command, so there's only one semicolon at the end of the command.

In this case, the width spec doesn't scale the image, it simply changes the size of the window through which the image is displayed. Since the car graphic is over 380 pixels wide, changing the width to 200 has the effect of cropping the image.

You can use triggers to run more than one action, so you can change the car image and change the specs for a block of text all at once. This time, let's suppose you want to use a rectangle as a button.

1. At the bottom of the *08-1_HideAndSeek* example, draw a rectangle. Name it *btnBar*.
2. Right-click (Control-click) btnBar and choose Open Actions for "btnBar" from the shortcut menu.
3. Choose the *click* trigger.
4. In the btnBar Actions panel, write these lines of code:

```
sym.$("car").animate(
{
    opacity: .25,
    width: 200
}, 2000
);
sym.$("tbHide").animate(
{
    fontSize: '12',
    fontFamily: 'Times'
}, 2000
);
```

If you're familiar with CSS, you may remember that you specify font size using a hyphenated identifier: *font-size*. Typefaces are also specified with a hyphenated identifier: *font-family*. In JavaScript, these identifiers don't use a hyphen, which might be confused with a subtraction operator. Instead they use what's known as camel text, where the first word begins with a lowercase letter and subsequent words start with an uppercase letter, as in *fontSize* and *fontFamily*.

You can specify the delay for *animate()* effects as you do with *show()* and *hide()* functions. In this example, the delay is set to 2000 for both the car image and the *tbHide* text box. The 2000 appears right before the closing parenthesis, and it is also separated by a comma.

Swapping Images in Edge

One of the keystones of web design is the image swap. You see it used all over the place, and JavaScript is usually the engine that's working behind the scenes. In the previous example, you saw how JavaScript could rewrite CSS specifications for formatting. In the case of an image swap, JavaScript is rewriting the *src* (source attribute) for an image tag. Want to make a button change its appearance when the cursor is over it? Use the *mouseover* trigger. You'll need a *mouseout* trigger to change it to its normal state when the cursor moves away. Want a button to change when it's clicked? You can use *mousedown* and *mouseup* to change its appearance. In each case, you want to swap the source of the button image. The perfect place to practice your image swapping chops is a photo gallery.

Photo Gallery Revisited

One common technique for displaying a group of photos is to have one large main photo and smaller thumbnail images, as shown in *Figure 9-5*. Your audience can click or mouse over a thumbnail to display that pic as the main photo. This type of a gallery works for up to about a dozen photos. If you try to use too many photos, the thumbnail images are too small to be useful. The project *08-3_Rollover_Gallery.zip* is ready to go. If you want to see the end result, check out *08-4_Rollover_Gallery_done.zip*.

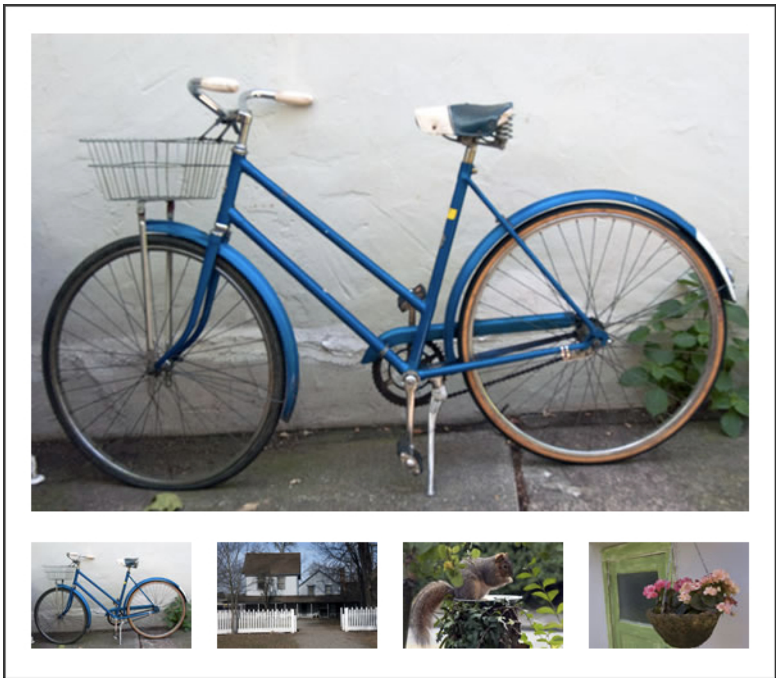


FIGURE 9-5

Here's a common layout for a rollover photo gallery. Point to one of the images at the bottom, and it is displayed as the main photo at the top. This technique uses simple image swapping code and works on computers, tablets, and phones.

Examine the gallery files and you'll find that some of the work has been done for you. The main photo and thumbnails are in place on the stage. In the Elements panel, you have four thumbnail images, each beginning with *sm_* (think "small"). The main photo is named "bike." Select it, and you'll notice the filename listed in Properties. All the images, big and small, are in the *images* subfolder of your project. In fact, there are three other images that don't appear on the stage: farmhouse, squirrel, and flowers. The project will use these photos, but they don't need to be imported as long as they're in the *images* subfolder. They'll be referenced as sources for the main photo. Follow these steps to finish up the rollover gallery:

1. Select the large bike photo. Then, in Properties, change the div tag to *img*.

   Edge automatically applies the <div> tag to everything you import. In this case, you want to use the src attribute of an <img> tag.

2. With bike still selected, in Properties change the image name to *mainPhoto*.

   In the Elements panel, you see mainPhoto when the image is selected. The bike won't remain a bike for long, so it's best to give it a more generic name that will make sense in your code.

3. Right-click (Control-click) the farmhouse thumbnail and choose Open Actions for sm_farmhouse, and then choose *mouseover* from the triggers.

   You're making life easy for gallery visitors. They won't even have to swap photos; all they have to do is move the cursor over a thumbnail.

4. On the left side of the Actions panel, click the Hide Element button.

   You don't really want to hide anything, but this is an easy, typo-free way to write much of the code you need. In the next step, you'll make alterations.

5. Change text1 to *mainPhoto*. Then change the *hide()* function to *attr()*.

   With these changes, your code should look like:

   ```
   sym.$("mainPhoto").attr();
   ```

6. Inside the parentheses for the *attr()* function, type: *'src','images/farmhouse.jpg'*.

   Now the line of code looks like this:

   ```
   sym.$("mainPhoto").attr('src','images/farmhouse.jpg');
   ```

   Make sure you use single quote marks around the src and photo attributes.

   The src attribute identifies the source for an image. The *images/farmhouse.jpg* text provides the path and name of the photo. You use similar code for each of the thumbnails.

7. On a new line in the Actions panel, type this line:

   ```
   sym.$("sm_farmhouse").css('cursor','pointer');
   ```

As explained in the box on page 175, this line will change the cursor to that familiar pointing finger when it is over the farmhouse thumbnail. The box also explains how to use the cursor button in Properties to create this effect.

8. Click the X button in the upper-right corner of the Actions panel. Go to File→Save and then press Ctrl+Enter (⌘-Return).

   The Actions panel closes, and you preview your project in your browser.

9. Move your mouse over the farmhouse thumbnail.

   The cursor changes to the pointing finger, and the main photo changes to show the farmhouse. If your project behaves differently, go back and double-check your code. Pay particular attention to the parentheses and those single quote marks. Make sure the photo path references the *images* folder, not the *image* folder.

10. Open the Actions panel for *sm_farmhouse*. Select and copy (Ctrl+C or ⌘-C) the code in the panel.

    You need to add similar code to the other three thumbnails. You might as well save time by copying, pasting, and editing the code.

11. Open one of the other thumbnails. Choose the *mouseover* trigger, and then paste (Ctrl+V or ⌘-V) in the code.

    When you paste it in, the code looks like this:

    ```
    sym.$("mainPhoto").attr('src','images/farmhouse.jpg');
    sym.$("sm_farmhouse").css('cursor','pointer');
    ```

    The text that needs to be edited to match the thumbnail is shown in bold.

12. Repeat the copy-and-paste process for the rest of the thumbnails.

When you're done, you have a working photo gallery. The thumbnails are big enough that they should work for an iPhone as well as a computer. If you want to dress the project up a little more, you can create rollover highlights or a drop shadow to mark the currently selected pictures.

OTHER USES FOR ROLLOVERS

All you need to do is roam the Web a bit, and you'll see all sorts of rollover examples, like changing the cursor to a pointing finger. They give your audience useful feedback. If you tackled the previous exercise, then you can see how you'd create a simple rollover button. Create two images for your button, one for its normal state and one for the mouseover state. As with the gallery example, give your images helpful names when you create them. With buttons, you may want to add something to the end of the name that identifies the state. For example, if you're creating website navigation buttons, you could have image filenames like *home.jpg, home_ovr.jpg, contact.jpg,* and *contact_ovr.jpg*. Import the normal state button to your project, and make sure the mouseover state button is in your images file, and you're ready to swap.

Identifying and Changing Elements and Symbols

When you're working with triggers and actions, half the battle is identifying the element that you want to change. The other half of the battle is changing its properties. When learning new coding techniques, it's best to start simple and build your skills. Consider a composition with two elements: a rectangle and a text box. In this case, the text acts as a button changing the width of the rectangle. Here's the lay of the land, as shown in *Figure 9-6*:

- The rectangle's ID is Square.
- The text box's ID is tbChangeSquare.
- There's nothing else on the stage.

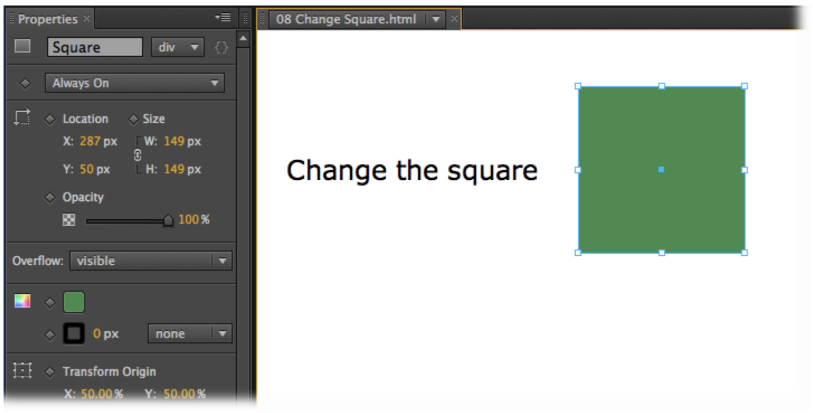


FIGURE 9-6

The stage is set for your first identification project. The rectangle's ID is set to Square because that's its shape to start. The text box's ID is set to tbChangeSquare. It will act as a button that changes the width property of Square.

You want to put your code in the "click" trigger for the text box tbChangeSquare. (For a refresher on using a click trigger see page 171.) Your code is like building blocks that you use to identify an element and its properties. You almost always start at the outermost level—that's the main stage. As explained back on page 170, you use the word "sym" to identify the stage.

```
sym.
```

Edge, like JavaScript, uses dot (.) notation to separate elements that are inside of other elements. Here the stage (sym) is one element and Square is another. You also use dots to separate the properties and methods that belong to elements. Now that the stage is identified, it's time to zero in on the rectangle with the ID "Square."

```
sym.$("Square").
```

Think back on page 173, during the discussion of show() and hide() functions, where the car element is identified using the jQuery selector. That's what's going on here.

The dollar sign ($) along with the ID is used to zero in on a specific element. We're going to use CSS to change the format of this element, so the next step is to add another building block:

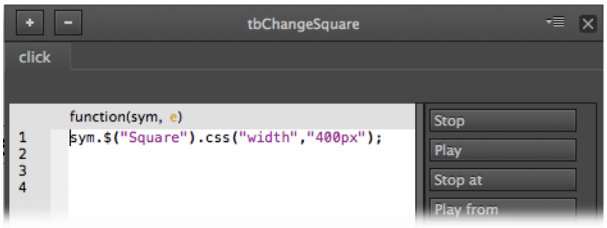
```
sym.$("Square").css()
```

A number of properties can be changed using CSS and, as explained on page 185, the process involves pairs: a property name and a value. These pairs go inside the parentheses, with each listed inside quotes. The name/value pairs are separated by a comma. Suppose you want to change the width of Square to 400 pixels: the final bit of code looks like this:

```
sym.$("Square").css("width","400px");
```

The units for width and height values are given in pixels (px). Lastly, you need to end the statement with a semicolon.

Right-click (Control-click) the text box tbChangeSquare and choose the click trigger. Paste or type your line of code and then test your project. It should look like *Figure 9-7*:



```
function(sym, e)
sym.$("Square").css("width","400px");
```

FIGURE 9-7

Here's the code that gets attached to the "click" trigger of the text box ID'd as tbChangeSquare. When the text gets clicked Square will be square no more.

Assigning Variables

It's common to use variables to identify different symbols, elements or values. In fact, that's what many of Edge's action buttons do. Look for ones with names like Get Symbol or Get Element. You get to choose the variable name, and then you assign a value to the variable. Suppose you want to shorten the way you identify Square. A line of code like this does the trick:

```
var theSquare=sym.$("Square");
```

Now you can use the variable theSquare to reference the element with the ID of Square. No more need to mention the stage (sym) or use the jQuery selector with dollar signs, parentheses, quotes and brackets. It gives you an easier way to make changes to that element. So, you can change the width, height, and fill color with commands like:

```
var theSquare = sym.$("Square");

theSquare.css("width","400px");
theSquare.css("height","400px");

theSquare.css("backgroundColor","blue");
```

But there's an even better way to change multiple properties of a single element. You don't have to repeatedly identify Square. You can identify it once and change multiple properties at once as long as the changes appear within curly braces. It works like this:

```
var theSquare = sym.$("Square");

theSquare.css({

    "width" : "400px",

    "height" : "400px",

    "backgroundColor": "blue"

});
```

Put curly brackets inside of the parentheses, then you can list several properties within those brackets. As usual, you need to put both the property name and the value in quotes. In this case, separate property names and values with a colon, then separate each property/value pair using commas.

Using Actions to Assign Variables

The Actions panel gives you snippets of code that can be used to assign symbols and elements to variables. For example, when you click on Get Element, Edge provides this snippet:

```
var element = sym.$("Text1");
```

As always, you can change the code for your own purposes, replacing "element" with any variable name you want. In this case, sym identifies the stage level element. Then replace Text1 with the ID of the child element you want to identify.

Click the Get Symbol button to add this snippet:

```
var mySymbolObject = sym.getSymbol("Symbol1");
```

In this case, the getSymbol() function replaces the jQuery selector, but really the drill is the same. Change the variable name (mySymbolObject) to something meaningful for your project. Then swap "Symbol1" with the ID of a symbol that's on stage or listed in your Element panel. Remember, there may be many instances on stage of a single symbol in the Library. When you're animating, you'll want to identify specific instances on the stage. The getSymbol() method identifies a specific instance of a symbol by its ID, which you'll find in the Properties panel when it's selected. You can also see the instance names in the Elements panel whether the instance is visible on stage or not. Once you've identified a specific instance of a symbol, you have access to its timeline and any child elements within the symbol.

TIP For more details on working with the getSymbol method, go Help→Edge JavaScript API (application programming interface). This opens Adobe's technical notes on the using JavaScript with edge in your web browser. Search for "getSymbol" and you'll find a section called Working with Symbols.

Checking Values with Text Boxes and Alert Dialogs

At this time Edge doesn't have much of a debugging toolbox, so you're on your own to devise some. When you begin assigning values to element properties and variables, you may want to check them. One quick and dirty way to do that is to create a temporary text box and then place the property or variable in question in the box. For example, perhaps you'd like to check the width of the variable theSquare.

1. Add a new text box to your project and change its ID to tbValue.

   Initially, you need to have some placeholder text in the box. Something like "Values" will work.

2. Open the click trigger that changes the size of theSquare.

   In the previous exercise, tbChangeSquare had the click trigger.

3. Add this line below the other code:

   ```
   sym.$("tbValue").html(theSquare.css("width"));
   ```

   Here's the breakdown of that new line: sym identifies the main stage and timeline. `$("tbValue")` identifies the text box. `html` is a property of textboxes and it expects text—a string of characters.

   Now, with a few explanatory comments, the code for the click trigger looks like this:

   ```
   // Assign the Square element to the variable theSquare

   var theSquare = sym.$("Square");

   // Change the width, height and backgroundColor properties of theSquare
   theSquare.css({
       "width"           : "400px",
       "height           : "400px",
       "backgroundColor" : "blue"

   })

   // Show the "width" value of theSquare in the text box tbValue
   sym.$("tbValue").html(theSquare.css("width"));
   ```

   There are some new things going on in that last line of code. The *.html()* method is used to add html code to a web page. (See the box on page 187 for more details on adding hyperlink tags to your text boxes.) Put a string of text inside of those parentheses and it appears on the page. In this case, it appears within the tbValue text box. You could put a string literal in the text box like this:

```
sym.$("tbValue").html('My manual is missing');
```

If you want to see the value of a property, all you need to do is put the property name within the parentheses. So, to display the theSquare's width you put theSquare("width") inside of the parentheses.

4. Test your composition.

   When you click on "Change the square," all the width changes take place and the tbValue text box displays the value "400px." If your composition doesn't work as expected, double-check the parentheses. In particular, make sure the last line ends with two closing parentheses and a semicolon.

TIP If you think your value-checking text box will be getting a lot of use, you probably want to assign it to a variable, too. For example:

```
var valueBox=sym.$("tbValue");
```

Then you can check values in this manner:

```
valueBox.html(theSquare.css("width"));
```

Using variables often simplifies the identification of elements. It not only speeds things up when you're coding, it means you're less likely to create a script-stopping typo.

USING *ALERT()* DIALOG BOXES TO CHECK VALUES

If you don't want to clutter up the stage with temporary text boxes, you can use the alert() dialog box in a similar fashion. It works in a browser like a pop up window. The code for an alert box looks like this:

```
alert("text appears here");
```

TIP If you place an alert() function in your code and the dialog box never appears, that's a sign that there's an error somewhere. Often, the error occurs before the alert() and so the script stops and the alert() function never runs.

The text within the quotes is displayed literally. However, you can use it just like the html property of the text box to display values of properties or variables. For example, you can do this:

```
alert(theSquare.css("backgroundColor"));
```

The result is that the background color of theSquare is displayed in the alert dialog box. You used the shorthand "blue" to describe the background color. JavaScript prefers to refer to the color's RGB values, so you see: rgb(0, 0, 255).

TIP If you're ready to move up to more powerful debugging tools for your JavaScript/jQuery projects, consider Firebug (http://getfirebug.com) an extension for the Firefox browser. Firebug, itself, has many extensions for different scripting languages and librarires. If you use Google's Chrome browser, check out the built-in Developer Tools. You can find them by clicking the wrench icon in the upper right corner, then choosing Tools→Developer Tools.

Identifying Elements within Symbols

When you start putting elements inside of symbols, you add another layer of complexity to identifying parts of your composition. Just remember to start on the outside and work your way in. Think of onions or those wooden Russian dolls nested inside each other. Suppose you have a symbol instance on the stage and an element inside of that symbol. The order to identify that interior element is: *stage* then *symbol instance* then *element*.

For experimentation, draw a square, a circle and a text box with "Words" shown. See *Figure 9-8*. Give them these IDs:

- Square ID = Square
- Circle ID = Circle
- Text Box ID = Words

Select all three elements and press Ctrl+Y (⌘-Y), the Modify→Convert to Symbol command. In the Library, name your symbol *BoxSymbol*. Drag two instances of BoxSymbol on to the stage. They'll need better unique names than the ones Edge provides. Give them IDs of *BoxInstanceOne* and *BoxInstanceTwo*.

For your first step, create a variable that identifies the instance called BoxInstanceTwo:

```
var boxTwo = sym.getSymbol("BoxInstanceTwo");
```

If you click the Get Symbol button, as explained on page 183, you need to replace Symbol1 with the ID of the symbol instance. You may also want to change the variable name mySymbolObject to something more helpful. The next step is to identify the element within the symbol. Suppose you want to change the text in the Words text box in boxTwo. You can zero in on the text box like this:

```
boxTwo.$("Words")
```

Next on the to do list is identifying the property for your text: html.

```
boxTwo.$("Words").html();
```

Your text can be a string literal (as explained on page 166) or it can be a variable that has a string assigned to it. A string literal might look like this:

```
boxTwo.$("Words").html("To be or not to be");
```

If you're going to use a variable, it needs to be defined in advance. Like this:

```
var Hamlet = "To be or not to be";

boxTwo.$("Words").html(Hamlet);
```

If you want to change the height property of the element with the ID of Square, the code would look like this:

```
var boxTwo = sym.getSymbol("BoxInstanceTwo");

boxTwo.$("Square").css("height", "400px");
```

If you want to trim down the code even more you can create a variable that identifies Square within the symbol:

```
var boxTwo= sym.getSymbol("BoxInstanceTwo");

var theSquare = boxTwo.$("Square");

theSquare.css("height","400px");
```

The first line is identical to the previous code. The second line creates a new variable called "theSquare" and assigns the element inside of boxTwo with the ID "Square" to the variable. The last line uses "theSquare" to identify the element within the symbol and assigns a value to the height property of the element. Even if there are several instances of the symbol on the stage, only the Square inside of boxTwo will change.

CODER'S CLINIC

Adding HTML Anchor Tags to Text Boxes

Back on page 141, you saw that it's easy to apply an HTML anchor tag <a> (sometimes called a hyperlink) to an entire text box, but not so easy to apply a link to specific words within a text box. If you've delved into the HTML for a web page, you've undoubtedly seen anchor tags like:

```
Why not buy a <a href=
"http://missingmanuals.com">Missing
Manual</a>?
```

In this example, the words "Missing Manual" appear on the web page as a hyperlink that opens the page identified by the URL within the quotes.

If you enter a line like that in a text box, you end up with everything. The tags which should be hidden, are visible and there's no hyperlink. The solution lies in the .html() method that works with text boxes. The exercise on page 184 shows how to use the .html() method to display text or the value of a variable. The main purpose of the .html() method is to add HTML code to a web page. Just place the code you want inside of the parentheses and inside of quotation marks. So that line shown earlier might look like this if it were placed inside of an Edge action:

```
sym.$("tbText").html("Why not buy a <a
href='http://missingmanuals.com'>Missing
Manual</a>?");
```

You can use all your favorite tags, like <h1>, <p>, <br/> and of course <a>. Squeeze as much as you want inside of those parentheses and your box will display it all.

The big question is, what do you want to use to trigger the html() method. You can use any trigger you want, but in most cases you won't use a "click" trigger to change big chunks of text on a page. More often, you'll want to use the stage's compositionReady trigger.

You can explore some of these symbol and element techniques in the completed example called *08-5_Get_Symbol_done*. See *Figure 9-8*. It shows how to change specific elements inside of two instances of the same symbol.

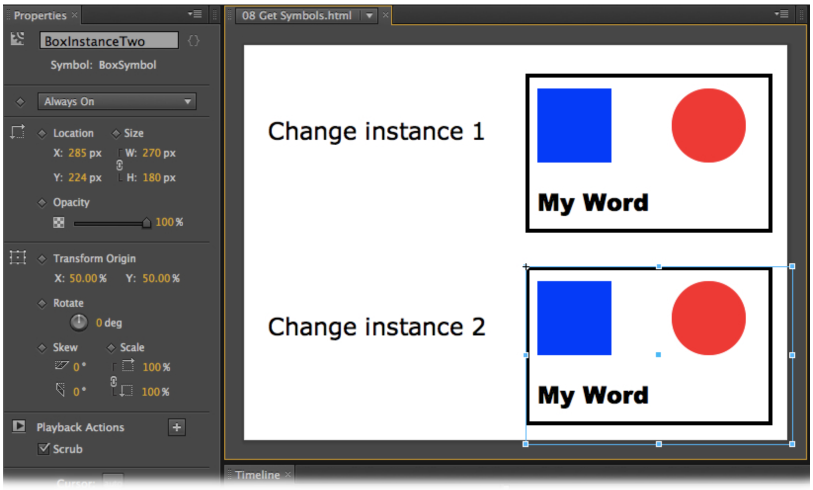


FIGURE 9-8

This composition has two instances of a single symbol. The text boxes have click triggers that change the colors of the square and circle and the text in the text boxes. Because these are unique instances, each symbol can change independently.

Playing a Symbol's Timeline

One of the great features of Edge symbols is that they have their own timeline, just like the main stage. You can start and stop that timeline using triggers, actions or your own custom code. Here's a project called *08-7_Control_Symbol_Timeline*. As shown in *Figure 9-9*, it has one symbol that acts as a simple counter. There are two text boxes: tbStop and tbPlay. The text serves as buttons to stop and start the timeline in the counter. If you've followed along since page 126, the technique for identifying the symbol with the ID myCounter should look familiar. The methods stop() and play() are part of all symbols. That includes the main stage and its timeline and the timeline of any symbol that you create. Here's the code in the click action for tbStop:

```
var mySymbolObject = sym.getSymbol("myCounter");

mySymbolObject.stop();
```

You can create that first line using the Get Symbol button in the actions panel. Just replace Symbol1 with myCounter, the ID for the symbol in your project. The code for tbPlay shouldn't come as a surprise:

```
var mySymbolObject = sym.getSymbol("myCounter");

mySymbolObject.stop();
```

In this case, the object with the trigger (the text box) is not part of the symbol, so it is necessary to identify the stage (sym) and the symbol you want to control (my-Counter). A variable (mySymbolObject) is created to easily identify the symbol. Then, the stop() and play() methods are used to control the timeline. If the object with the trigger were part of the symbol, or if everything were on the stage, you can use sym to identify the current symbol. Then the code to stop and play the timeline would be:

```
sym.stop();

sym.play();
```

Check the actions for the red and green blocks in the symbol, and you find exactly that code. You can also use some of the tricks you learned on page 129, to jump to a particular point on the timeline. For example, you can make the timeline Play from a particular point in time:

```
var mySymbolObject = sym.getSymbol("myCounter");

mySymbolObject.play(4000);
```

The basic unit of time is the millisecond, so 4000 is the point four seconds into the timeline. As an alternative, you can use a timeline label as described on page 107:

```
var mySymbolObject = sym.getSymbol("myCounter");

mySymbolObject.play("myLabel");
```

Getting the Current Playhead Position

Sometimes you'll want to know the current position of the playhead in a symbol or in your main timeline. Edge provides a function that does just that: getPosition(). It returns the number in Edge's favorite unit of time, the millisecond, where 1000 equals a second. Suppose you have a variable for a symbol that is named theCounterSymbol. A statement that assigns the current playhead position to a variable looks like this:

```
var playPosition = theCounterSymbol.getPosition();
```

What can you do when you know the playhead position? For one thing, with a little arithmetic you could make the playhead jump ahead a couple of seconds.

```
var playPosition = theCounterSymbol.getPosition();

playPosition = playPosition + 2000;

theCounterSymbol.play(playPosition);
```

Line two adds 2000 to playPosition and then assigns the result to playPosition, giving it a new value that's two seconds farther down the timeline. Want to go backwards? Just use subtraction. The third line uses playPosition with its new value to reposition the timeline playhead.

Using Conditional Statements

There's one problem with the previous exercise that positions the playhead on the timeline. If the current position is too far along the timeline, the updated value for the variable playPosition may be greater than the duration of the symbol. That means you want to check the value and then perform different actions based on the result. That's the perfect job for the *if* conditional statement described on page 168. The basic form is this:

```
if (this statement is true) {

    do these actions;

    }

    else {

    do these other actions;

    }
```

The parentheses and curly brackets are all vitally important. Forget one, and your carefully constructed conditional won't work. The else portion of the statement is optional, but you need it for the current dilemma. Here's a statement that checks to see playPosition is under the ten-second mark. If it is, the playhead is moved to playPosition. If not (the else portion of the statement), the playhead is moved to 0, the start of the timeline.

```
if (playPosition < 10000) {

    theCounterSymbol.play(playPosition);

    }

    else {

    theCounterSymbol.play(0);

    }
```

Attach this code to the click trigger of a forward pointing arrow, and you have a way to jump ahead in your animaton. With a little tweaking, you can create a back arrow and code that moves in the other direction. If you'd like to examine this code and some of the other symbol controls, take a look at *Figure 9-9*.

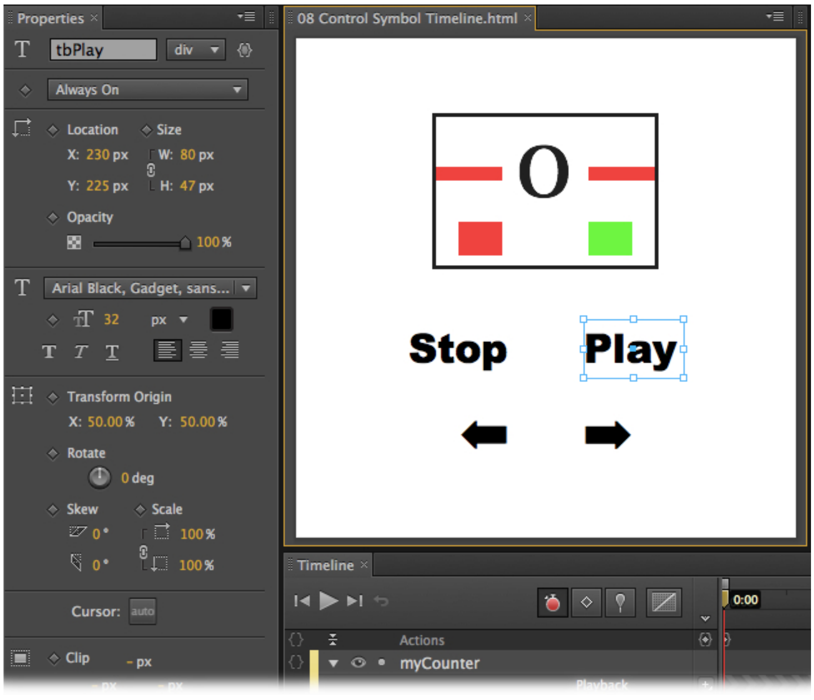


FIGURE 9-9

Edge provides several methods for playing and stopping the timeline. These methods work for symbols as well as the main timeline. To see the code for the examples described in this section, get 08-7_Control_Symbol_Timeline from the Missing CD at http://missingmanuals.com/cds/edgepv5mm.

PART

4

Appendix

APPENDIX:

Installation and Help

Installation and Help

As of this writing, Edge isn't yet a commercial project. Adobe offers a free preview so that web builders can put Edge through its paces and provide feedback for the team that's building the Edge of the future. In spite of its immaturity, Edge, as it is now, can help you build some great animated features for your web pages. One of the things you might find lacking is help and support. This appendix is here to help you get Edge installed and to point out some places where you can find some help. After all, even Lewis and Clark had guides to help them in their explorations.

NOTE This section was based on Adobe Edge Preview 5.1. Your mileage may vary if you're using a more recent version.

Edge System Requirements

The Preview version of Edge runs in Windows 7 and Mac OS 10.6. That means Windows XP and Mac OS 10.5 and below are not supported. For those of you who are interested in such details, Edge is a 32-bit application but will run on 64-bit machines, too. Adobe has not listed any other requirements, but Edge doesn't seem to require a lot of processing power or disk space. If your system can comfortably run a web browser and another application, like a photo editing program, at the same time, you won't have trouble with Edge. The Windows application folder requires less than 100 megabytes. The Mac version comes in under 120 megabytes.

Installing Edge

Installation of the Preview version of Edge is fairly easy. The first step is to download the file for your computer from Adobe's website. There's a version for Windows and a version for Macs. You need to have an Adobe account to log in and get access to the files. If you don't already have an account, don't worry. It's free. All you have to do is provide the usual information: name, email, and so forth.

You can find the download links at: *http://labs.adobe.com/technologies/edge/*. The Windows version is a ZIP file. The Mac version is a DMG file.

Installing Edge for Windows:

1. Double-click the ZIP file. Extract the contents of the ZIP file to the desktop.
2. Open the folder containing the extracted files, and double-click *Set-up.exe*.

   The Welcome screen appears.
3. Click Accept to start the installation.

   The Install Options screen opens.
4. Click Install to Install Edge or Back to return to the Welcome screen.

   The Installation Progress screen appears; when installation is complete, the Thank You screen is presented.
5. Click Done to close the installer or the Adobe Edge button to launch Edge.

Installing Edge for Macs:

1. Double-click the DMG file.
2. Open the Adobe Edge Preview folder and double-click the Install application.

   The Welcome screen opens.
3. Click Accept to start the installation.

   The Install Options screen appears.
4. Click Install to Install Edge or Back to return to the Welcome screen.
5. The Installation Progress screen appears; when installation is complete, the Thank You screen is presented.
6. Click Done to close the installer or the Adobe Edge button to launch Edge.

Uninstalling Edge

When you're working with Preview versions of a product like Edge, that means sooner or later you're going to want to uninstall one version and install the latest, greatest version. Often, it's important to make sure you uninstall completely, or the next version won't behave well.

Uninstalling Edge for Windows:

1. Select Start→Control Panel and, in the Programs section, click "Uninstall a Program."
2. In the list of programs, double-click Adobe Edge.

NOTE The Adobe Edge program entry has no icon in the Add/Remove programs screen.

   The Uninstall Options screen appears.

3. Click Uninstall and wait while Windows deletes Edge.

   To back out now and keep Edge around for a while, click Cancel.

Uninstalling Edge for Macs:

1. In the Applications→Adobe Edge Preview folder, double-click the Edge Uninstaller application.
2. Click Uninstall and wait while your Mac deletes Edge.

   To back out now and keep Edge around for a while, click Cancel.

TIP Sometimes Adobe applications leave remnants on your hard drive even when they've been uninstalled. That can be a particular problem with "preview" applications. The remnants interfere when you install a new version of the same application. If you have these kinds of problems you can use Adobe's "cleaner" tool to remove those remnants. Follow Adobe's instructions carefully when using the cleaner utility. You can find the tool and the instructions at *www.adobe.com/support/contact/cscleanertool.html*.

Getting Help

In the Preview versions of Edge, if you click on the Help menu, you won't find the usual help system. No 500-page, PDF-style user's manual. No context-sensitive help items. The fact of the matter is, they're probably still writing the documentation. In Preview 5, there is a working document called Adobe Edge Runtime API. It consists of a couple of pages of fairly technical details about how Edge works. There are a couple of tips for writing JavaScript code. If you're new to Edge or JavaScript, it's probably not going to provide a lot of help.

If you have a question, you're better off turning to the Edge community forum that's hosted on Adobe's website: *http://forums.adobe.com/community/labs/edge/*. You need to use your Adobe ID to log in, but you already have that if you downloaded Edge. What you'll find in the forum is an active community of Edge fans. Other Edge explorers, like you, post some of their projects and share info about how they work. Post a question, and soon you'll have an answer—if not a full-blown debate.

Books for HTML, CSS, JavaScript, and jQuery

A few of the books in the Missing Manual series cover HTML, JavaScript, and jQuery: *HTML5: The Missing Manual* by Matthew MacDonald (O'Reilly); *CSS: The Missing Manual* by David Sawyer McFarland (O'Reilly); *JavaScript and jQuery: The Missing Manual* by David Sawyer McFarland (O'Reilly); and *Creating Websites: The Missing Manual* by Matthew MacDonald (O'Reilly).

Head First jQuery by Ryan Benedetti and Ronan Cranley (O'Reilly) also serves up a great introduction to jQuery coding. If you're looking for the Bible on JavaScript, turn to *JavaScript: The Definitive Guide* by David Flanagan. Just be prepared for some serious technical detail.

Online Help

There are a number of non-Adobe resources for help with Edge, JavaScript, and jQuery. Do a Google search for "Adobe Edge," and you're likely to find websites where people post examples and maintain blogs that discuss Edge. Darrell Heath, one of the technical reviewers for this book, has just such a site at: *www.heathrowe.com*.

For help with JavaScript, you can turn to *www.w3schools.com/js* or *https://developer.mozilla.org/en/JavaScript.*

The official jQuery website provides tutorials, forums, and lots of ways to learn. Here's a link to its tutorials: *http://docs.jquery.com/Tutorials*.

Index