Beginning JavaScript for Adobe Acrobat®

A non-programmer's guide

John Deubert

QED Guides
Quality Electronic Documentation
from Acumen Training

Dedication

For (in order of descending height) Barbara, Elizabeth, Gigi, and Julia.

Beginning JavaScript for Adobe Acrobat
John Deubert

ISBN-13: 978-0-9850512-0-4
ISBN-10: 0-985-05120-5

version 1.0 (Sample)

To report errors, send a note to errata@acumentraining.com

Notice of Rights

Notice of Liability

The information in this book is distributed "as is," without warranty. While every precaution has been taken in the preparation of the book, the author shall not have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indireclty by the instructions contained in this book or by the computer software and hardwre products described in it.

Trademarks

Trademarks are used throughout this book. Rather than put a trademark symbol in every occurrence of a trademarked name, we state that we are using the names in an editorial fashion only and to the benefit of the trasdemark owner with no intention of infringement of th trademark.

Adobe Acrobat is a trademark of Adobe Systems Incorporated.

Table of Contents

Only the links to the first two chapters work in this sampler.

Chapter 0
Introduction

Welcome to Acrobat JavaScript.

You are about to open a new chapter in your work with Acrobat forms. In reading this book, you will learn how to add features and abilities to your forms that are not otherwise possible: roll-over help, automatic text field formatting, database connectivity, and more will all become routine parts of your form design. You will also learn a programming language, perhaps your first: JavaScript. Even if you've never had a hankering to write software and sling code, you'll find JavaScript an important arrow in your quiver; it's supremely useful, relatively easy, and surprisingly fun.

Of course, you don't need JavaScript skill to create an Acrobat form; you've no doubt been doing perfectly well for quite some time without it. However, a knowledge of JavaScript adds immeasurably to your ability to make your forms look and behave exactly as you want. JavaScript will open to you a world of possibility whose scope is hard to overstate. Sufficient to say that the knowledge and techniques you will learn in this book will allow your forms to take on a sophistication that is otherwise completely impossible.

If this sounds like hyperbole, think again. It's true.

This Book

What this book is

This e-book is a non-programmer's guide to the JavaScript programming language as used in Adobe Acrobat; it teaches you, step-by-step, how to add specific features to your Acrobat forms and, in so doing, teaches you about the JavaScript language.

If you are an experienced Acrobat form designer, but have never written a line of programming code in your life (and pretty much wanted to keep it that way), then this book is written for you. Together we'll explore programming concepts while learning how to add features to your Acrobat forms: we discuss arrays while creating a dynamic list of prices; we talk about **`case`** statements while creating a pop-up menu; we teach a form to check the version of Acrobat on which it's running and, along the way, talk about **`if`**–**`else`** constructs.

When you're done with the book, you will be in good shape to read more formal books on JavaScript and to make use of Adobe's technical specification of Acrobat JavaScript.

About the Book

What this book isn't

This book is *not* a complete reference to the JavaScript language or to using JavaScript in Acrobat. JavaScript is a broad and deep language and is capable of much that we don't discuss here.

The purpose of this book is to bootstrap you to a point where you can learn the rest of the language's abilities on your own; it assumes you are an experienced Acrobat forms designer, but have little or no experience with computer programming languages. If you are already comfortable with Objective C, Java, or other programming language, this book will be paced too slowly for you; you should go directly to Adobe's Acrobat JavaScript Object Specification.

Also, this book does not teach you how to make form fields and other components of Acrobat forms; I assume you have reasonable experience with the mechanics of designing and creating forms.

Mac or Windows?

With one exception the examples in this book will work with either the Macintosh or Windows version of Adobe Acrobat. The exception is the pair of chapters on Acrobat database connectivity, which is available only in Windows. Otherwise, the Mac and Windows versions of Acrobat work identically, except for minor differences dialog box labelling, etc. Illustrations in this book are taken from both versions of Acrobat.

How to use this book

The first two chapters of this book present basic information and terminology and must be read before attempting the rest. The remaining chapters are probably best read in order, since learning a programming language is inevitably a cumulative activity. Nonetheless, Chapters 3-19 are designed so that one can usefully read them in any order; if you need to add roll-over help to a form, feel free to skip to Chapter 7 and see how to do it. Just don't be surprised if you find yourself having to go back to earlier chapters to clarify references to mysterious terminology.

Some chapters need to be read in order; for example, the discussion of regular expressions is spread across Chapters 10, 11, and 12 and they must be read in that order to be sensible at all. Most chapters, however, are intended to be semi-independent.

All of the chapters, however, assume you have read Chapters 1 and 2.

About the Book

Reading the book on-screen

Being an e-book, *Beginning JavaScript for Adobe Acrobat* is distributed electronically; its pages are designed to be easily read on-screen. Since it's distributed as a self-contained PDF file, you can read the book on nearly any computer or tablet. It turns out that the page size works particularly well on a 768x1024 screen, such as that of the iPad; that's probably a coincidence.

The PDF file has been "enabled" for commenting in Adobe Reader, so you should be able to make notes to yourself on the pages in either the full Acrobat or Reader.

URL links within the book are all "live" and will open the appropriate page in your default web browser.

Printing the book

Many people (myself included) prefer to read technical tutorials on paper. To that end, feel free to print *Beginning JavaScript;* it works very well printed two-up and double-sided (**Figure 0.1**).

Sharing the book

I'd rather you didn't, really. *Beginning JavaScipt* is reasonably priced and I'm hoping to write additional books on this and other topics. However, I do need to be able to make at least a minimal living at it. So, if your friends or colleagues want to read the book, encourage them to buy their very own copy. They'll feel good about themselves.

Registering the book

E-books are similar to software; in particular, it is possible for an author to update an e-book, improving explanations, fixing the grammar, killing typos, and eliminating flat-out errors. (This particular e-book, of course, has neither errors nor tpyos.)

If you register your book purchase (go to www.acumentraining.com/qedguides/qedregister.html), you'll be notified whenever there is a new update for your book. It's worth doing.

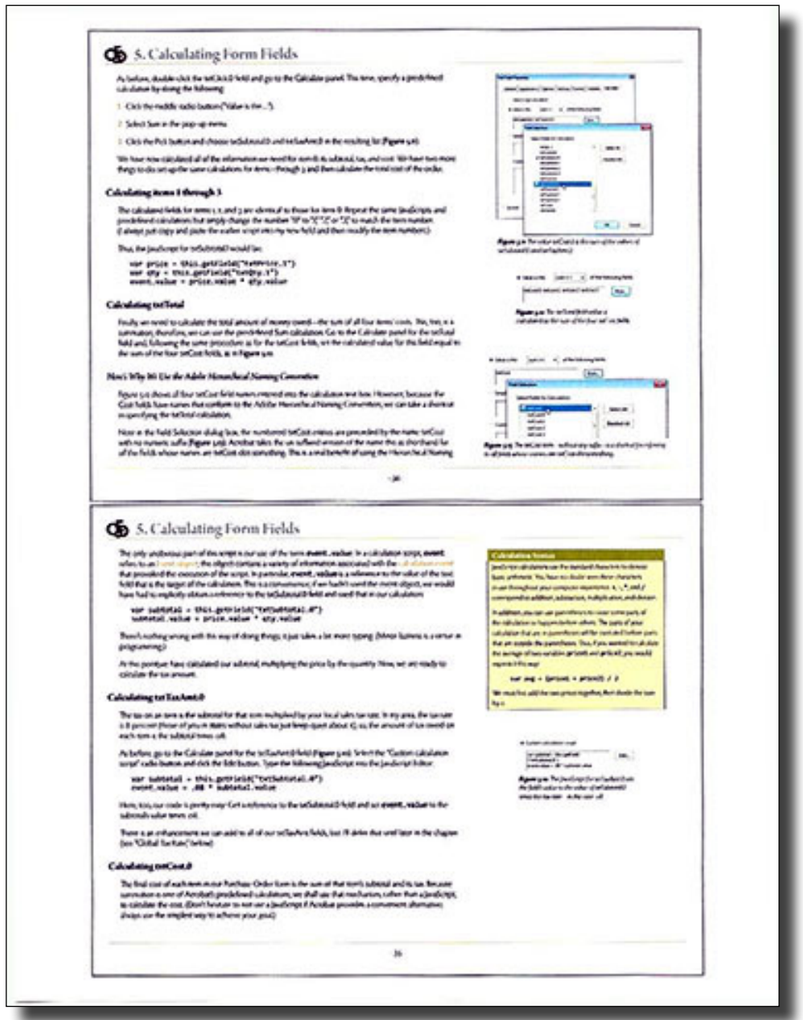


Figure 0.1 *This book's page size works well printed two-up on A4 or American Letter-size paper.*

Online Resources

There are two sources of information that are specifically intended to be used with this book.

Sample Files and Support

Each chapter in this book is built around one or two sample forms to which we add JavaScripts to complete the chapter's goals. Each sample form has two files associated with it:

- The complete, functioning form, with all JavaScripts in place. This shows you how the form is supposed to behave and allows you to inspect the chapter's JavaScripts in place. This version of the form has a filename ending with *"end."*
- A version of the form with all form fields and other elements in place, but no JavaScripts attached. Use this version if you want to follow along while reading the chapter, writing, debugging, and executing the JavaScripts as you go. The file for this version of the form will have a name ending in *"start."*

The sample form files are available for downloading at www.acumentraining.com/qedguides/acrojs.html. This web page is also where you will find errata, update information, and other useful information.

Acumen Journal

The *Acumen Journal* (**Figure 0.2**) is a free periodical that I produce three or four times per year. Each issue has an article on advanced Acrobat usage and many of these are about JavaScript. The back issues make a good supplemental to the topics in this book; there are more than 60 issues accumulated since I began writing the *Journal* back in 2000. You can download them at www.acumentraining.com/acumenjournal.html.

If you wish to be notified when a new issue of the *Journal* comes out, there is a link on the Acumen Journal web page to an appropriate form.

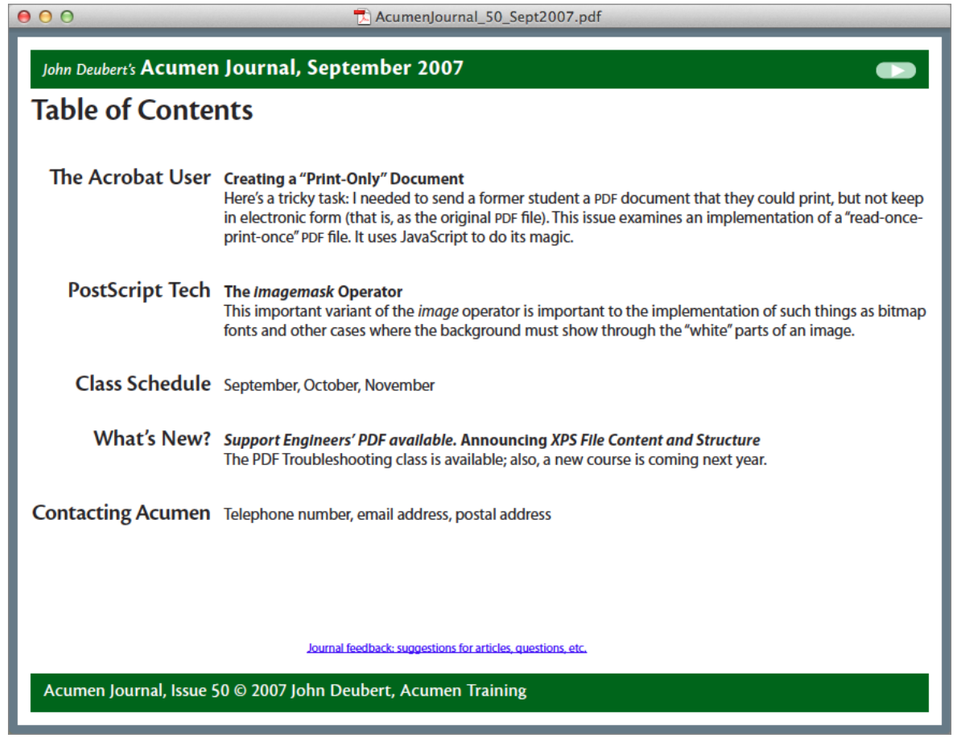


Figure 0.2 *Each issue of the Acumen Journal has an Acrobat-related article; many of these are about JavaScript usage.*

Thank *You*

Finally, thank you for buying this book or looking at the sample chapters. Either way, you are participating in an ongoing experiment in publishing.

Chapter 1
Welcome to JavaScript

Life is full of threshold phenomena.

Periodically in life, you learn something that broadens your world immeasurably, revealing an expanse of new experience, problems, opportunities, play, and work. Whole worlds that had invisibly surrounded you suddenly appear, providing a new space to explore. Reading, sex, driving, children, all bring with them concerns, interests, and interactions that had been previously inaccessible and unsuspected.

In its own small way, learning JavaScript will be just such a threshold event in your professional life. If you've been working with Acrobat for any length of time, you've probably gotten pretty good at it and have become quite comfortable at creating forms, adding music, creating slide shows, and all the other features Acrobat offers.

This book introduces the New World. A knowledge of JavaScript allows you to do things within Acrobat that far exceed what you've done so far: You can interface with databases; add your own pop-up menus; create forms with sophisticated, interactive interfaces; and implement form fields that can look up prices and other data. These are only a few of the things you can do within your Acrobat documents using JavaScript. The extent to which you can manipulate your Acrobat files is vastly greater with JavaScript skills than without.

Hence, this book.

Here you will learn how to accomplish a variety of useful tasks in Acrobat using JavaScript. Along the way, you will learn a great deal about JavaScript, programming, and Adobe Acrobat.

What We'll Learn in this Chapter

In this chapter, we'll learn:

- What's a JavaScript?
- Types of JavaScript
- Attaching a JavaScript to a form field
- JavaScript objects, data types, and syntax
- JavaScript errors and the debugger
- How to use your own text editor with Acrobat JavaScripts

What You Should Know Already

This book assumes you have reasonably extensive experience in working with Acrobat and creating Acrobat forms. In particular, I assume you know how to create forms in Acrobat; you should be able to create a form field, set its properties, and assign actions to it. If you feel vaguely uneasy about any of these tasks, you may want to run right off and buy a book on the subject (see the sidebar on the next page).

Beyond that, this book does not assume any knowledge of programming; you will learn the programming skills you need as we proceed through our examples.

Again, this book is intended for programming novices; if you are an experienced programmer, you may it to be paced slower than you'd like. Experienced programmers may do better to just go to a more-advanced

book to learn JavaScript (there are myriad such books, though they all teach JavaScript in the context of web pages) and then look at Adobe's JavaScript reference site to see how to apply it to Acrobat.

What Version of Acrobat Should You Have?

Although this e-book presumes you are working with Acrobat X, the current version, the instructions we step through will nearly all apply to Acrobat 9, as well. Where there is a large difference between the two versions, I'll provide a note on what you should do in Acrobat 9.

What *Is* JavaScript?

JavaScript is a programming language. The term "programming language" often induces jitters in newcomers, but, conceptually, it's not very scary: A programming language is a language that is used to describe the steps involved in carrying out some task. In Acrobat, these tasks include moving to a particular page of a document, sending data to a database, and calculating a form field value. Carrying out the steps described by a JavaScript is referred to as executing the program.

As a programming language, JavaScript's most significant characteristic is that it's simple enough that many applications use it as their native scripting language. Web browsers can all interpret JavaScripts embedded in Web pages, and, particular to our topic, Acrobat can execute JavaScripts attached to form fields, pages, and PDF files.

Like any language, JavaScript has its own vocabulary (words that have meaning) and syntax (rules by which you make statements with those words). Learning JavaScript, therefore, has much in common with learning a human language, such as Spanish or German, only it's much easier. JavaScript is vastly simpler than any human language: there are no metaphors, no literally nonsensical idioms, no synonyms, no subtle shades of meaning. Just very precise statements telling Acrobat to do something very specific.

JavaScript in Acrobat

Acrobat allows you to create four different kinds of JavaScripts:

- *Form Field JavaScripts* are attached to form fields. Acrobat executes the script when a particular event occurs in that form field, such as a button click. Most JavaScripts in Acrobat are attached to form fields.
- *Page JavaScripts* are executed when the user moves to or leaves a particular page in the Acrobat document.
- *Document JavaScripts* are executed when the Acrobat Document opens.
- *Document Action JavaScripts* are executed user opens, closes, saves, or prints a document.

Books on Acrobat Forms

Although there are a number of books that will teach you how to create Acrobat forms., most of them are out of print. The only extant book is:

- ***PDF Forms Using Acrobat and LiveCycle Designer Bible***
  Ted Padova and Angie Okamoto

  Ted Padova is the king of the "Bible" style books and this volume shows why.

There are other books specific to Acrobat forms that, though out of print, are still available if you look around. In particular, my own old book is still available, though it's getting pretty long in the tooth, I must admit:

- ***Creating Forms in Adobe Acrobat*** – John Deubert

Finally, any reasonably complete book on Adobe Acrobat will have at least a chapter or two on creating forms. This includes my own book:

- ***Adobe Acrobat X: Visual Quickstart Guide*** – John Deubert
- ***Acrobat X Classroom in a Book*** – Adobe Creative Team
- ***Acrobat X PDF Bible*** – Ted Padova

These are all available through Amazon.com, as is pretty much everything in the world.

We shall talk about Page, Document, and Document Action JavaScripts in Chapter 2. For now, let's look at how you type in and use a Form Field JavaScript.

Our First JavaScript

Let's start exploring our new world by adding a simple JavaScript to the button in **Figure 1.1**.

This form consists of a set of flash cards that are intended to be printed double-sided and then used to quiz students on vocabulary terms. Our PDF file has only a few flash cards; each card has a button that takes users to an order form they can use to purchase the complete set of cards. We are going to add a JavaScript to the Order Form button that takes the user to the order form, located on the last page of the Acrobat file (**Figure 1.2**).

As will be true throughout this book, there are two versions of this form on the *JavaScript for Acrobat* web page:

- *JSAcro_Ch01_Example_Final.pdf* is the full form, complete with all relevant JavaScripts.
- *JSAcro_Ch01_Example_Start.pdf*, the "raw" version, lacks the chapter's JavaScripts so that you may type in the JavaScript yourself if you wish.

These are available at www.acumentraining.com/qedguides/acrojs.html.

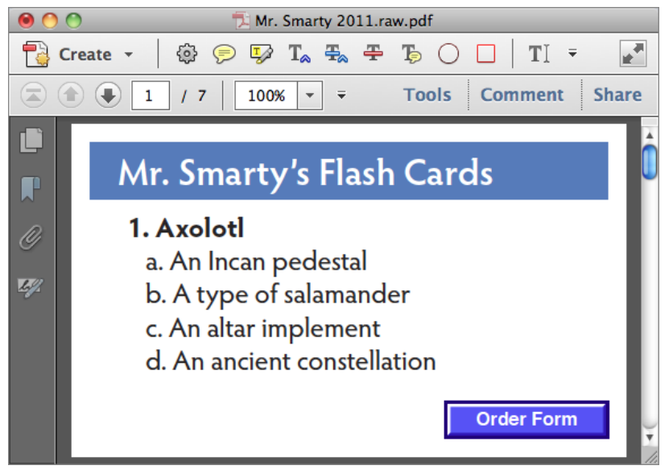


Figure 1.1 *We're going to add a JavaScript to the "Order Form" button in this form.*

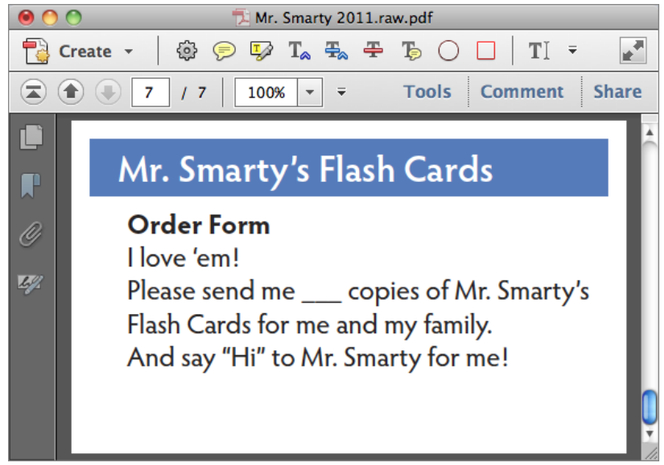


Figure 1.2 *The Order Form button's JavaScript will move the user to the PDF file's final page, which is the order form.*

Where Do the Sample Files Live?

Remember that all of the sample files in this eBook are available at **www.acumentraining.com/qedguides/acrojs.html.**

Attaching a JavaScript to a Form Field

As a reminder of something you may already know, let's step through the process of attaching a JavaScript to a button, in this case our Order Form button.

To attach a JavaScript to a button:

Start with the form open in Adobe Acrobat and the Tools pane exposed.

> ***Note***
> Acrobat 9 doesn't have a Tools pane, but it *does* have a Forms menu. To get to the Form Editor, select *Forms > Add or Edit Fields.* Having done this, you can follow the steps exactly as listed.

1 In the Tools pane's Forms panel, click on the Edit tool (**Figure 1.3**).

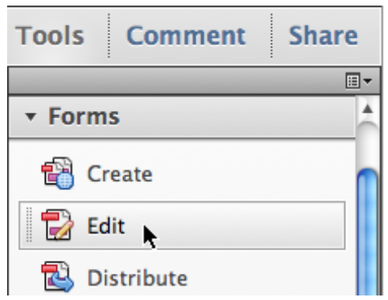


Figure 1.3 *Enter Form Editor mode by clicking the Edit button in the Forms panel.*

Acrobat will enter Form Editor mode, displaying all of the form fields on the current page as a set of rectangles and replacing the usual three Tasks panes with a single Forms pane (**Figure 1.4**). Note in the figure that our form has only one field on its first page, a pushbutton field with the name *btnOrderForm.*

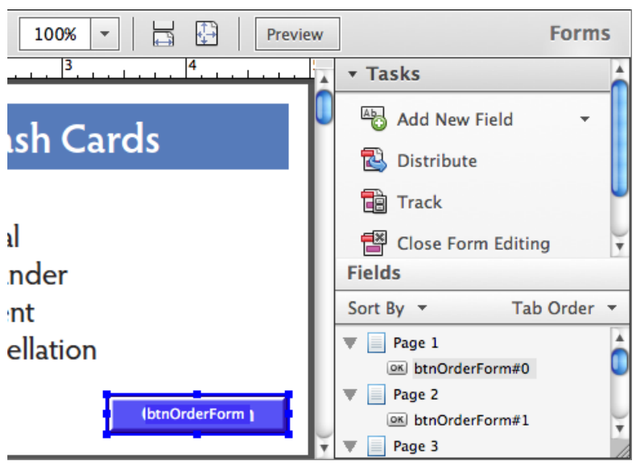


Figure 1.4 *Acrobat's Form Editor mode gives you a Forms pane, new toolbars, and presents each form field as a rectangle with handles..*

2 Double-click the Order Form button.

Acrobat will present you with the Field Properties dialog box (**Figure 1.5**).

3 Click on the Actions tab.

You will now be looking at the set of controls that specify what should happen when you click on this button (**Figure 1.6**).

4 In the Select Trigger pop-up menu, select *Mouse Up*.

This tells Acrobat that this button's action should trigger when the mouse button is released after clicking in the button. (See the sidebar, at right.)

5 In the Select Action pop-up menu, select *Run a JavaScript* and then click the Add button.

Acrobat will present you with the JavaScript Editor, a dialog box with a simple text editor (**Figure 1.7 ,** next page). This is where you type in the JavaScript that should be associated with the Order Form button.

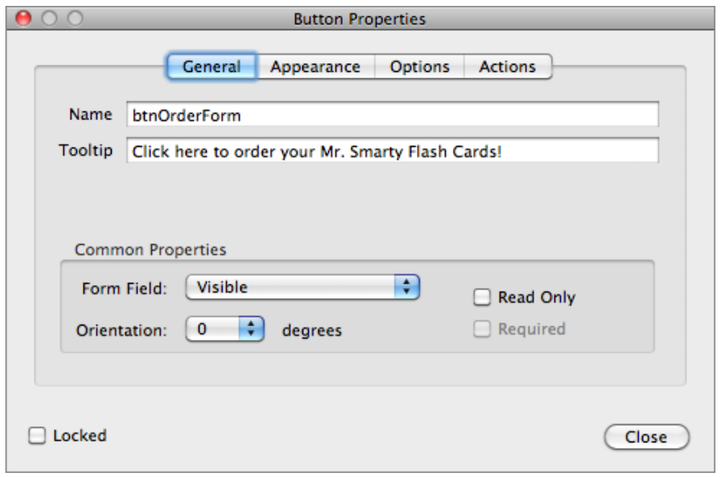


Figure 1.5 *Double-clicking on the button yields the Button Properties dialog box.*

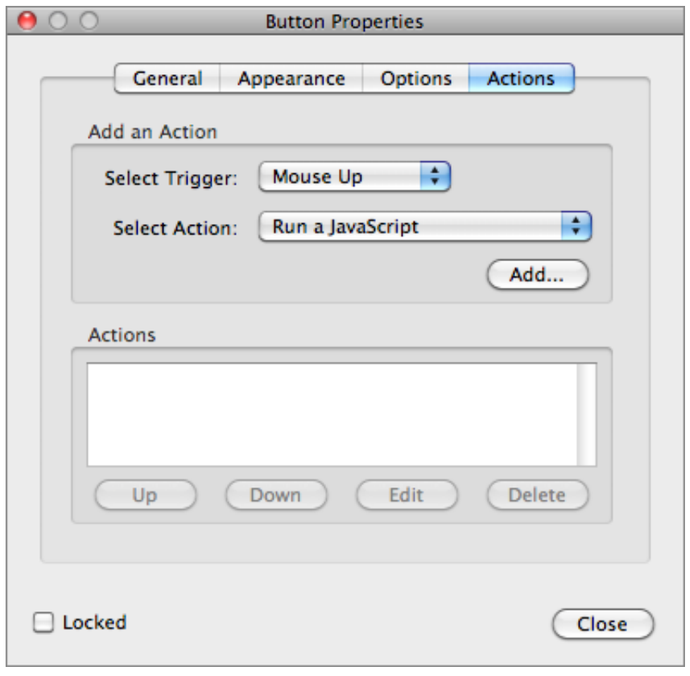


Figure 1.6 *The Actions tab of the Button Properties dialog box is where we specify what should happen when the user clicks our button.*

6 Type your JavaScript into the text field of this dialog box.

In the case of our order form, the JavaScript is a two-line program that moves the user's view of the document to the page containing the order form and then causes the user's computer to beep:

```
this.pageNum = 6
app.beep
```

Type these lines into the text-editing field exactly as above, making sure to match upper and lowercase. The first line of code says, "In this document, set the current page number to 6." The second line tells the Acrobat application to beep.

7 Click the OK button of the JavaScript Editor and the Close button of the Field Properties dialog box to return to the Acrobat form.

You are now looking at the Acrobat flash card page with the Form tool still selected, as in Figure 1.4.

8 Exit the Form Editor by clicking the Close Form Editing button in the Forms pane (**Figure 1.8**).

You are now back where you started, looking at your document page.

So, now try it out. Click the Order Form button, and Acrobat will move to the order form page and then beep.

Form Field Events

The Select Trigger pop-up menu, shown in Figure 1.6, offers six form field events to which you can attach an action:

- **Mouse Down** occurs when the user presses the mouse button with the pointer in the form field.
- *Mouse Up* occurs when the user clicks on the field and then releases the mouse button with the mouse pointer still in the field.
- *Mouse Enter* occurs when the mouse pointer first rolls over the form field.
- *Mouse Exit* occurs when the mouse pointer rolls out of the form field.
- *On Focus* occurs when the user clicks on or tabs into the form field, so that it becomes the target for keyboard or other input.
- *On Blur* occurs when the user tabs out of a form field or clicks on some other form field, so that our field is no longer the target for user input. (Blur is the opposite of Focus, of course.)

The Field Properties dialog box, Figure 1.6, lets you associate one or more Actions (a JavaScript action, in our case) with any of these events.

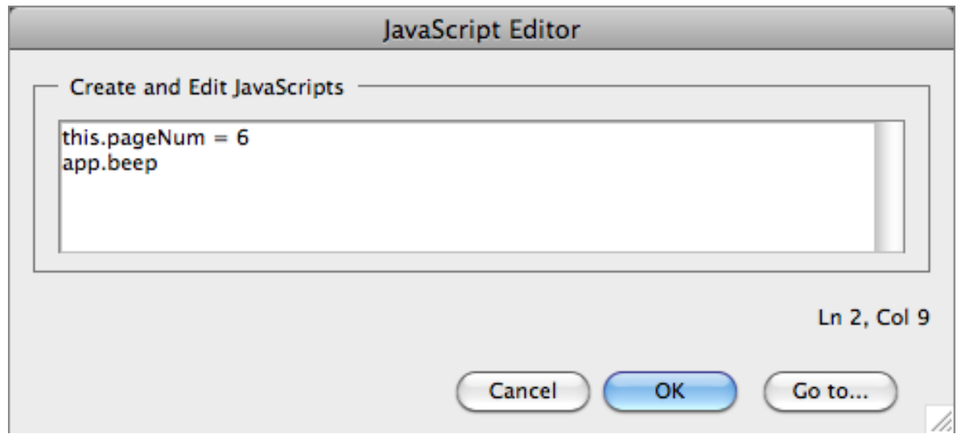


Figure 1.7 *The JavaScript Editor window is a simple text editor that you will use to type in your JavaScripts.*

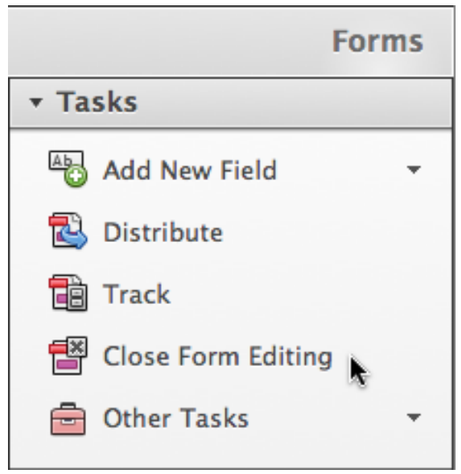


Figure 1.8 *You exit the Form Editor by clicking Close Form Editing in the Forms pane.*

JavaScript Objects

Our two-line JavaScript makes use of two JavaScript objects. A JavaScript object is the representation of some piece of data within your JavaScript program. Before your program can manipulate or examine a form field, it must first create an object that represents that field. Most of the things you can manipulate in JavaScript (pages, signatures, database connections, and so on) are represented in your program as objects.

In our sample program, `this` refers to a Doc object. A `Doc` object represents an open Acrobat file to your JavaScript program; you use this object to change pages, save the document, and otherwise manipulate the

document from within your program. The word **this** in our sample JavaScript refers particularly to the Acrobat document in which our JavaScript resides (the Flash Cards file, in our case); think of it as short for *this document.*

The word **app** is an App object, a reference to the Acrobat application being used to view the current document. You use an **App** object to tell the Acrobat application to do something: open a file, put up an alert dialog box, or, in our case, beep.

Commonly used JavaScript object types include:

- *Annot* represents an annotation (for example, a "sticky note") in the current document.
- *App* represents the Acrobat application being used to view the current document.
- *Connection* represents a connection to an external database.
- *Doc* represents an open Acrobat document.
- *Field* represents a form field.
- *Sound* represents a sound embedded in the current document.

Object properties

JavaScript objects are analogous to physical objects in the world around us, such as books, vases, and dogs. Every real-world object possesses a set of characteristics that define it (such as, for a dog, color, tail length, and number of fleas).

The characteristics of a JavaScript object are referred to as that object's properties. These are elements of an object that our JavaScript programs can examine and change as needed. Each type of object has a set of properties that characterize it; for example, Doc objects have, among other things, a title, a current page number, an author, and a number of pages (see **Table 1.1**).

Table 1.1 Document Object Properties (Partial)

Property	Data Type	Description
author	String	The person who wrote the document
fileSize	Integer	The size of the PDF file, in bytes
numPages	Integer	The number of pages in the document
pageNum	Integer	The page number currently visible to the user
title	String	The name of the document.

Data Types

The Data Type column of Table 1.1 lists the type of information associated with each of the properties it lists. Computer programming, including JavaScript, uses special terms to precisely describe types of data. Here are the terms commonly used in JavaScript:

- *Integer* is a whole number, such as 1, 2, 87, or -6293.
- *Floating Point* is a number with a fractional part, such as 1.7, -842.9011, 1024.0. Note that the fractional part may be zero, as in 1024.0; in this case, the floating point number has the same value as an integer, though internally it is still a floating point number. Floating point numbers are often referred to as "floats."
- *Boolean* is an entity that can have two values: true or false. Boolean data are used to describe characteristics that can have only two states. (For example, the *spayed* property in our dog object is a Boolean value; a dog either is or isn't.)
- *String* is text, that is, a "string of characters."

The phrase **this.pageNum** addresses the **pageNum** property of the Document object; this property is the page number the document is currently displaying to the user. Our program moves the user to the order form page by setting the current document's **pageNum** property to the order form's page number:

```
this.pageNum = 6
```

Some observations about this page number assignment:

- You address the property of an object by naming both the object and the property, joined by a period:

  ```
  object-name.property-name
  ```

- The equal sign in the line of code above is an assignment command; it sets the value of something. In our case, it sets the current document's page number to 6.
- JavaScript is case-sensitive. Upper- and lowercases are distinct; our program would have failed if we had typed **This.PageNum**.
- Acrobat internally numbers a document's pages starting at zero; the seven pages in our Acrobat file are numbered zero through six. Thus, when our JavaScript set the pageNum property to 6, it was moving us to the last page in the document.

"Program" vs. "Script" vs. "Code"

Here are three closely related terms that we'll be using throughout this book.

- A *program* is a general term for a series of instructions that tell a computer in detail how to carry out a particular task. In general, a program is a stand-alone set of instructions, such as an application.
- A *script* is a program that is intended to manipulate and run within another program. JavaScript is a scripting language, because you use it to control the behavior of another program, such as Acrobat or FireFox.
- *Code* is the term applied to the set of instructions that make up a program. Your JavaScript program is made up of JavaScript code.

Object methods

A method is a command that is associated with a JavaScript object. Just as a dog can be given commands ("Sit," "Heel," "Spit that out this instant!"), JavaScript objects have commands that they can carry out. The set of commands is different for each type of object. For example, **Table 1.2** lists some of the commands the **app** object knows how to execute.

Table 1.2 App Object Methods

Method	Arguments	Description
beep		Play the system's "beep" sound
alert	String	Put up an alert dialog box with the specified text
goBack		Go to the previous view
goForward		Go to the next view
newDoc		Open a new, blank Acrobat document
openDoc	String	Open an Acrobat file. The string argument contains the name of the file

In our Order Form JavaScript, we executed ("called") the **app** object's beep method:

```
app.beep
```

Note that we call an object's method in the same way that we refer to one of its properties: the object name, a "dot," and the method name.

Arguments and Return Values

Some methods need additional information in order to carry out their task; for example, the **openDoc** method listed in Table 1.2 needs to know the name of the file you want to open. Information handed to a method is called an argument to that method. The **openDoc** method takes a filename as its argument; this information, surrounded by parentheses, must follow the method name. An invocation of **openDoc** looks something like this:

```
app.openDoc("TermPaper.pdf")
```

The above JavaScript statement would open an Acrobat file named *TermPaper.pdf*.

Sometimes when you give a dog a command, you expect the dog to give you something back: the command, "Fetch the stick, boy!" should yield a stick in your hand (along with some gratuitous drool). Similarly, many JavaScript methods have a return value, some piece of data they give back to the JavaScript program. The **openDoc** method we invoked above actually returns a Doc object representing the newly opened document, though our single-line use of **openDoc** just ignores it. We shall look at return values in much more detail in the next chapter.

Named Arguments

Generally, you must supply arguments to a method in a certain order; the **app** object's **alert** method, which we'll talk about in detail in the next chapter, wants a string and an icon code:

```
app.alert("Woah! Somethin' weird just happened!",2)
```

The string and the integer must be supplied in that order so that JavaScript can identify them.

However, you can also pass arguments to a method by name. The Acrobat JavaScript Guide (described at the end of this chapter) defines a name associated with each of the arguments a method requires; in the case of **app.alert**, the names are cMsg and **nIcon**.

You can supply these arguments, in any order, using the following call to the method:

```
app.alert({cIcon: 2, cMsg:"Woah! Somethin' weird just happened" })
```

Note that we have braces within the method's parentheses and within those we have our method arguments. Each argument is represented by the name of the argument, a colon, and then the argument's value; the arguments are separated by commas.

This is not as concise as passing the arguments by position, but it is clearer as to the purpose of each of the arguments. (Our first call to **app.alert**, for example, gives no clue to the purpose of the 2.) Also, passing arguments by name give you great flexibility in formatting your code. In particular, you can place line breaks within the argument list and supply the arguments in any order; our previous example could have been written

```
app.alert({cIcon: 2,
           cMsg: "Woah! Somethin' weird just happened" })
```

In this book, we shall pass arguments by position; we'll use argument names only when it's necessary—usually for clarity—to a particular JavaScript example. I shall, however, supply the names of the arguments when describing methods so that you can use them if you wish.

Usually, there's no overwhelming reason to do so.

JavaScript Program Syntax

Here we must discuss a couple of short topics regarding how JavaScript commands are put together into a program.

JavaScript Statements

A JavaScript program—any computer program—consists of a series of statements, each of which carries out one step in the overall task. Our sample program consists of two statements: a page assignment and a call to the **app** object's beep method.

```
this.pageNum = 6
app.beep
```

Usually, each line within a JavaScript program will contain a single JavaScript statement, as in our program. You can put more than one statement on a line, separated by semicolons. Our two-line program could have been written on a single line:

```
this.pageNum = 6; app.beep
```

Why would you do this? Purely for esthetics; some people just prefer to combine very simple statements together. I recommend against this practice; most programs are much easier to read if you have only one statement per line.

If you read other people's JavaScripts, you may notice that many programmers put semicolons at the end of every line in their program:

```
this.pageNum = 6;
app.beep;
```

This doesn't hurt anything, but it's unnecessary. Most of them do it out of habit; JavaScript looks very much like the programming languages C and C++, both of which require that all statements end with semicolons. You can leave out the semicolons.

JavaScript Text

JavaScript programs are simply text files; you can write them with any text editor or word processor and then copy and paste them into the Acrobat JavaScript Editor dialog box. In fact, Acrobat lets you specify an external editor that should be used for editing your JavaScripts; we'll discuss how to do this at the end of the chapter.

By the way, space and tab characters within a JavaScript line have no particular meaning in JavaScript. You can use them as you wish to format your program. This is a purely visual issue; you want to format your JavaScript code so that it's easy to read.

Use whitespace characters lavishly! Reading program code is tedious at best; a program can be nearly undecipherable if the programmer has not formatted the code for easy reading. This is an important enough issue that I shall be providing formatting tips for many of the JavaScript constructs we use in this book.

JavaScript Comments

JavaScript code can be pretty cryptic. Puzzling over someone else's code (or even your own code from six months ago), trying to figure out exactly what it's trying to do, can be tedious. As a courtesy to others looking at your code and as an aid to your future self, it is very important to place comments in your JavaScript code.

A JavaScript comment is text in your code that is ignored by the JavaScript "machine." The purpose is to let you place your own notes to be read by human beings examining the code.

JavaScript recognizes (that is, ignores) two kinds of comments:

- *Single-line comments* start with a double slash (`//`) and extend to the end of the current line in the JavaScript code. These are intended for brief comments.

```
//This is a single-line comment.
```

- *Block comments* start with a `/*` and end with a `*/`. Between these two delimiters can be as much text as you wish, spread out over as many lines as you wish within the JavaScript file. Use this for longer comments.

```
/* Here we have a block comment.
This text will be completely ignored
until we end the comment, right here. */
```

An Example

Consider the following, uncommented JavaScript from later in this book:

```
var txtField = event.target
txtField.fillColor = color.red
txtField.textColor = color.white
```

Since we have not yet discussed these commands, exactly what the purpose of this script is and how it carries out that purpose is very unclear, although it does seem to have something to do with color.

On the other hand, if we include comments in the code, then it becomes possible for someone unfamiliar with the program to at least know what the intent of the program is and generally what it's doing:

```
/* This program changes the background and text color of
   a text field when the user tabs into or clicks in the
   field. */
var txtField = event.target      // Get a reference to the text field
txtField.fillColor = color.red   // Set the background to red
txtField.textColor = color.white // Set the text color to white
```

This version of the program is much clearer, even to someone new to the code.

Comments are a Force for Good in programming. Any script more complex than a couple of lines should include comments that describe what it does and how it works.

All of the examples in the rest of this book will be heavily commented to make them as comprehensible as possible.

You're welcome.

JavaScript Errors

In the (ahem) rare event that you have an error in your JavaScript—you misspelled a variable name, mis-copied a piece of code, and so on—you will be faced with the task of figuring out what is wrong with your code, a process known as debugging. Acrobat left to itself treats these errors quietly; if the code fails, Acrobat just aborts the script; to all appearances, clicking on the button did nothing at all, although there was really a failed script happening beneath the hood. This isn't useful for debugging; if a JavaScript fails, we'd like to know about it and, furthermore, be told what went wrong, so we can fix it.

To get this diagnostic information about failed scripts, we need to enable something called the JavaScript Debugger.

The JavaScript Debugger

The JavaScript Debugger is a window that presents information about JavaScripts executing in your Acrobat document (**Figure 1.9**). In particular, the Debugger presents diagnostic information about errors in your JavaScripts, allowing you to figure out why they are misbehaving. The Debugger also provides a variety of other tools useful in working with your scripts; we shall examine these tools in detail in a later chapter.

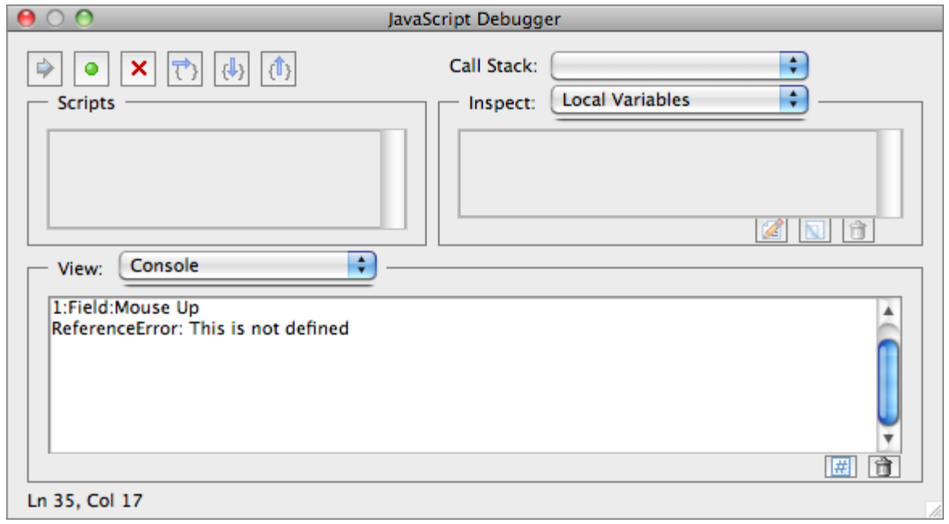


Figure 1.9 *The JavaScript Debugger window presents information about errors in your JavaScript, as well as providing a variety of other tools.*

With the JavaScript Debugger enabled, whenever one of your JavaScripts fails, Acrobat will open the Debugger with an error message ("ReferenceError," in Figure 1.9). This will allow you to determine what went wrong and what to do about it.

Error messages can be somewhat cryptic at first, but with time and familiarity they become useful. The most common messages you are likely to see are the following:

- `Reference Error: XXX is not defined`

  This indicates that you misspelled something; the name "xxx" (or whatever) is not one that JavaScript knows. Remember that JavaScript is case-sensitive; there is a difference between **app** (which JavaScript knows) and **App** (which it doesn't know).

- `Syntax Error`

  This means that JavaScript could not make sense of something in the code. This usually means you omitted something (a comma, a number, a parenthesis) from your script. An example of a syntax error would be

  ```
  app.alert("Hi, Mom", 3
  ```

  This line is missing its closing parenthesis.

The set of possible error messages is very large, though most are pretty hard to provoke. Just sit tight, read the message, and carefully examine the aberrant JavaScript code for misspellings and omissions.

Enabling the JavaScript Debugger

You enable the JavaScript debugger in the Preferences of your copy of Acrobat. Open your Acrobat preferences and select JavaScript in the long list of preference categories (**Figure 1.10**). All of the checkboxes in the JavaScript Debugger section should be selected, as in the figure.

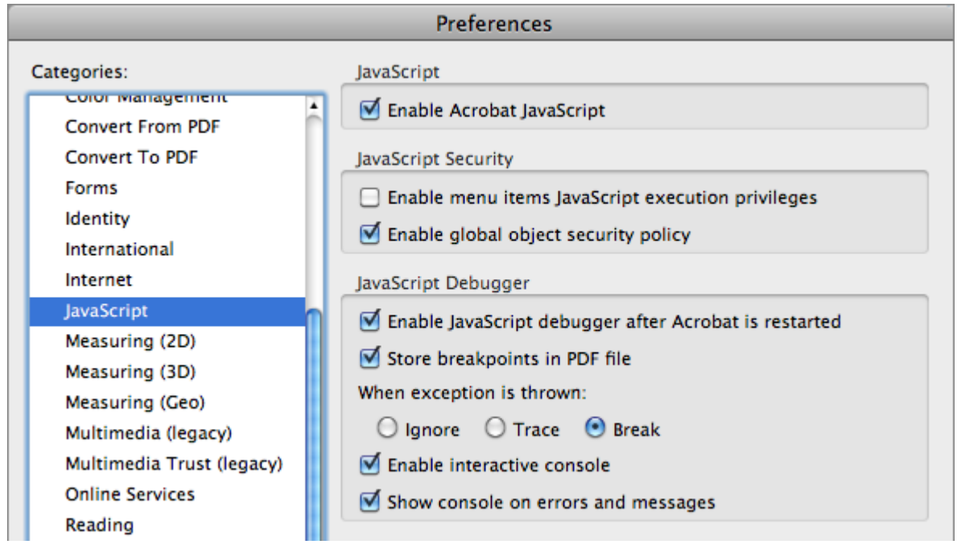


Figure 1.10 *You enable the JavaScript debugger in your Acrobat preferences. Select all of the checkboxes in the Debugger section, as above.*

Using Your Own Text Editor

The text editor built into Acrobat's JavaScript Editor (see Figure 1.7) is pretty minimal. It lets you type in your JavaScript, but it has no particularly fancy editing capabilities. For short JavaScripts, this is not important; when typing long, complex scripts, however, you will miss having a fully featured text editor.

You can ask Acrobat to use text editing software of your choice to edit your JavaScripts. Having told Acrobat what editor you want to use, it will automatically launch this software when you click on the Add or Edit button in the Document JavaScripts dialog box (**Figure 1.11**).

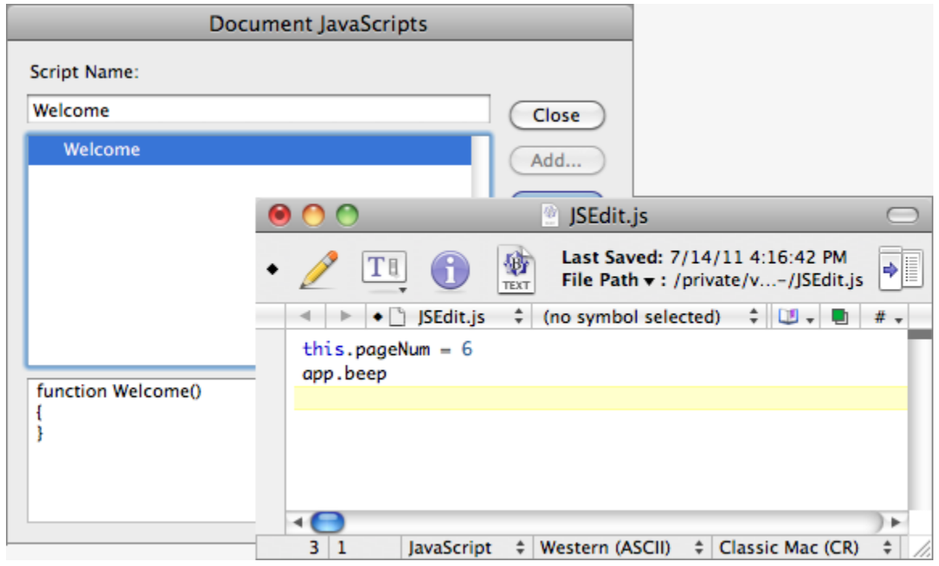


Figure 1.11 *Acrobat will launch your external text editor whenever you click the Ad or Edit button in the Document JavaScripts dialog box.*

To set this up, you must specify in Acrobat's Preferences the editor you wish to use for working with JavaScripts.

To specify a text editor to use when editing JavaScripts:

Start with Acrobat open.

1. On the Mac, select *Acrobat>Preferences;* on Windows, select *Edit>Preferences.*

   Acrobat will present you with its Preferences dialog box (**Figure 1.12,** next page).

2. Select JavaScript in the list of Preferences categories.

   The Preferences dialog box will display the controls that affect Acrobat's JavaScript support, as in Figure 1.12.

3. Among the JavaScript Editor controls, at the bottom of the dialog box, select Use External JavaScript Editor.

4. Click the Choose (Windows) or Browse (Mac) button and then navigate to the .exe or .app file for the editor you want to use when editing JavaScripts.

5. Click the OK button.

That's all there is to it. Now, when you edit a JavaScript, Acrobat will automatically launch your text editor. Type your JavaScript code into the text editor's window, save the text, and then close the editor. Your JavaScript will be automatically entered into Acrobat.

For what it's worth, my favorite programmers' editor on the Mac is TextWrangler (www.barebones.com); it is a first-rate editor and completely free, omigawd. Among Windows text editors, I'm rather fond of TextPad (www.textpad.com); it's relatively inexpensive shareware and well worth the money.

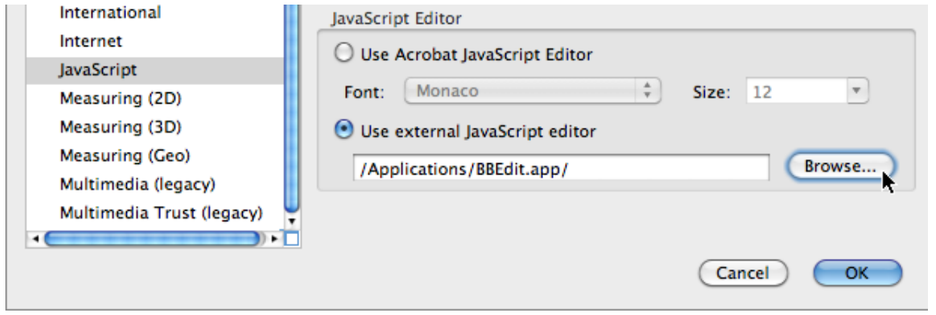


Figure 1.12 *You can tell Acrobat to use an external editor when typing or modifying JavaScript code.*

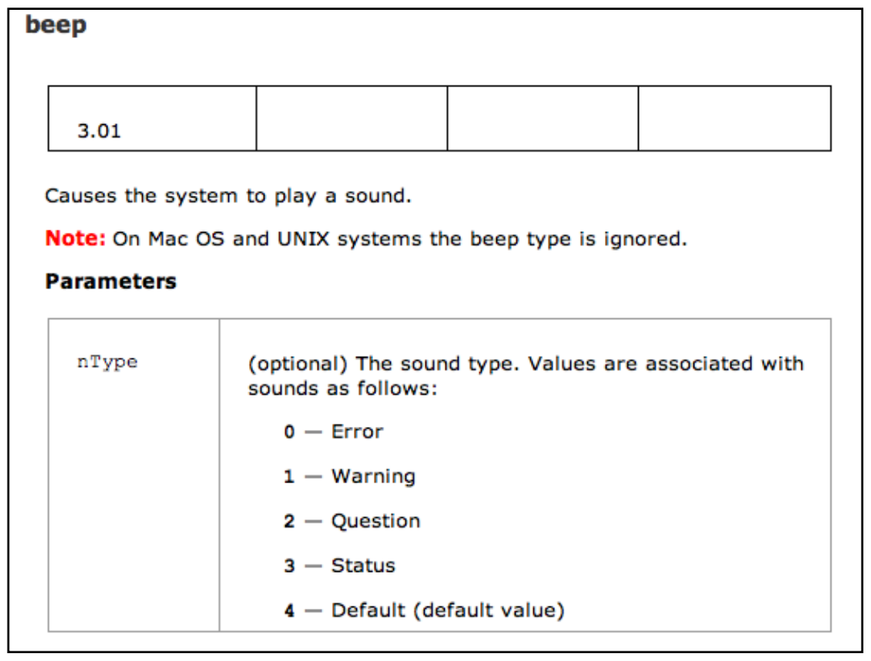
Acrobat JavaScript Guide

This book is a non-programmer's introduction to using JavaScript within Adobe Acrobat. The full description of all of the things you can do with JavaScript in Acrobat is presented in a document available from Adobe's Developer website: the *JavaScript for Acrobat API Reference* or *"JSAPI,"* from now on (**Figure 1.13**). This is the technical specification of all of the object types available to your JavaScript programs within Acrobat. There is also an excellent on-line version available through Adobe's Developer website.

The *JSAPI* is a technical specification, not a document you would willingly read from one end to the other. It gives a detailed description of every JavaScript object type available in Acrobat and the properties and methods of each. Where the book you are reading presents a series of examples of how to carry out specific tasks in JavaScript, the *JSAPI* describes *everything* you can do in Acrobat with JavaScript.

> ***Note***
> The JSAPI is available as a Kindle book from amazon.com.

To give you a bit of the flavor of the *JSAPI*, **Figure 1.14** shows a screenshot of the complete description of the **app** object's **beep** method. I shall be making occasional references to the *JSAPI* throughout this book.



beep

3.01			

Causes the system to play a sound.

Note: On Mac OS and UNIX systems the beep type is ignored.

Parameters

nType	(optional) The sound type. Values are associated with sounds as follows: 0 — Error 1 — Warning 2 — Question 3 — Status 4 — Default (default value)

Figure 1.14 *The description of the **app** object's **beep** method is a good example of the type of description provided for every object., method, and property in the JavaScript for Acrobat API Reference.*