

Table of Contents (Summary)

	Intro	xxiii
1	the interactive web: <i>Reacting to the Virtual World</i>	1
2	storing data: <i>Everything Has Its Place</i>	33
3	exploring the client: <i>Browser Spelunking</i>	85
4	decision making: <i>If There's a Fork in the Road, Take It</i>	135
5	looping: <i>At the Risk of Repeating Myself</i>	189
6	functions: <i>Reduce, Reuse, Recycle</i>	243
7	forms and validation: <i>Getting the User to Tell All</i>	289
8	wrangling the page: <i>Slicing and Dicing HTML with the DOM</i>	343
9	bringing data to life: <i>Objects as Frankendata</i>	393
10	creating custom objects: <i>Having It Your Way with Custom Objects</i>	449
11	kill bugs dead: <i>Good Scripts Gone Wrong</i>	485
12	dynamic data: <i>Touchy-Feely Web Applications</i>	537

Table of Contents (the real thing)

Intro

Your brain on JavaScript. You're sitting around trying to *learn* something, but your *brain* keeps telling you all that learning *isn't important*. Your brain's saying, "Better leave room for more important things, like which wild animals to avoid and whether naked water skiing is a bad idea." So how *do* you trick your brain into thinking that your life really depends on learning JavaScript?

Who is this book for?	xxiv
We know what you're thinking	xxv
Metacognition	xxvii
Bend your brain into submission	xxix
Read me	xxx
The technical review team	xxxii
Acknowledgments	xxxiii

the interactive web

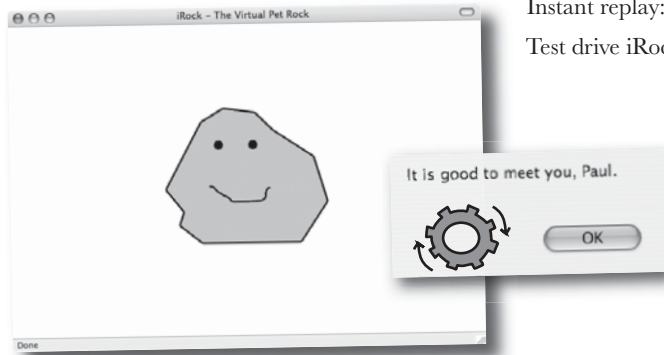
1

Reacting to the Virtual World

Tired of thinking of the Web in terms of passive pages?

Been there, done that. They're called books. And they're good for reading, learning... lots of good things. But they're not **interactive**. And neither is the Web without a little help from JavaScript. Sure, you can submit a form and maybe do a trick here and there with some crafty HTML and CSS coding, but you're really just playing *Weekend at Bernie's* propping up a lifeless web page. Real live **interactivity** requires **a bit more smarts** and a **little more work...** but it has a ***much bigger payoff***.

(Online) people have needs	2
Like talking to a brick wall... nothing happens	3
But JavaScript talks back	4
Lights, camera, interaction!	6
Use the <script> tag to tell the browser you're writing JavaScript	11
Your web browser can handle HTML, CSS, AND JavaScript	12
Man's virtual best friend... needs YOUR help	15
Making iRock interactive	16
Create the iRock web page	17
Test drive	17
JavaScript events: giving the iRock a voice	18
Alerting the user with a function	19
Add the iRock greeting	20
Now let's make the iRock really interactive	22
Interaction is TWO-way communication	23
Add a function to get the user's name	24
Instant replay: what just happened?	27
Test drive iRock 1.0	28



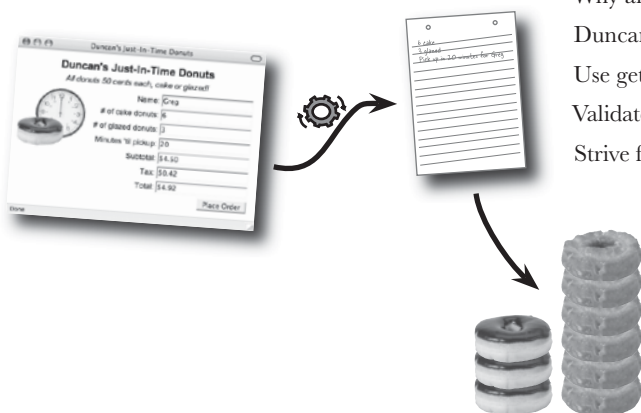
storing data

2

Everything Has Its Place

In the real world, people often overlook the importance of having a place to store all their stuff. Not so in JavaScript. You simply don't have the luxury of walk-in closets and three-car garages. In JavaScript, **everything has its place**, and it's your job to make sure of it. The issue is **data**—how to **represent it**, how to **store it**, and how to **find it** once you've put it somewhere. As a JavaScript storage specialist, you'll be able to take a cluttered room of JavaScript data and impose your will on it with a flurry of virtual labels and storage bins.

Your scripts can store data	34
Scripts think in data types	35
Constants stay the SAME, variables can CHANGE	40
Variables start out without a value	44
Initialize a variable with “=”	45
Constants are resistant to change	46
What's in a name?	50
Legal and illegal variable and constant names	51
Variable names often use CamelCase	52
Plan the Duncan's Donuts web page	56
A first take at the donut calculations	58
Initialize your data...or else	61
NaN is NOT a number	62
You can add more than numbers	64
parseInt() and parseFloat() convert text to a number	65
Why are extra donuts being ordered?	66
Duncan discovers donut espionage	70
Use getElementById() to grab form data	71
Validate the web form's data	72
Strive for intuitive user input	77



exploring the client

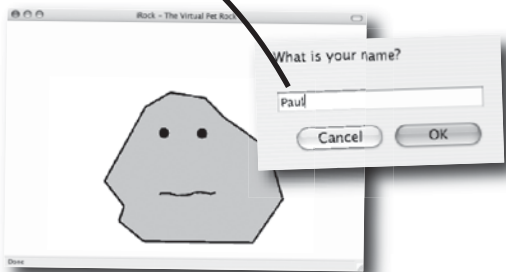
3

Browser Spelunking

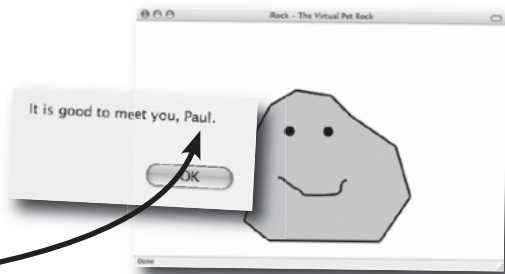
Sometimes JavaScript needs to know what's going on in the world around it. Your scripts may begin as code in web pages but they ultimately live in a world created by the browser, or client. **Smart scripts** often need to know more about the world they live in, in which case they can **communicate with the browser** to find out more about it. Whether it's finding out the screen size or accessing the browser's snooze button, scripts have an awful lot to gain by cultivating their browser relationship.

Clients, servers, and JavaScript	86
What can a browser do for you?	88
The iRock needs to be more responsive	90
Timers connect action to elapsed time	92
Breaking down a timer	93
Set a timer with setTimeout()	94
A closer look: the setTimeout() function	95
Multiple size screens, multiple complaints	99
Use the document object to get the client window's width	100
Use document object properties to set the client window width	101
Set the height and width of the iRock image	102
The iRock should be sized to the page	103
onresize is triggered when the browser's resized	107
The onresize event resizes the rock	108
Have we met? Recognizing the user	110
Every script has a life cycle	111
Cookies outlive your script's life cycle	112
Cookies have a name and store a value... and can expire	117
Your JavaScript can live OUTSIDE your web page	119
Greet the user with a cookie	120
greetUser() is cookie-powered now	121
Don't forget to set the cookie, too	122
Cookies affect browser security	124
A world without cookies	126
Talk to the users... it's better than nothing	129

Start here!



Finish!



decision making

If There's a Fork in the Road, Take It

4

Life is all about making decisions. Stop or go, shake or bake, plea bargain or go to trial... without the ability to make decisions, nothing would ever get done. It works the same in JavaScript—**decisions allow scripts to decide between different possible outcomes.** *Decision-making drives the “story” of your scripts*, and even the most mundane scripts involve a story of some sort. Do you trust what the user entered and book her a trip on a Sasquatch expedition or do you double-check that maybe she really just wanted to ride a bus to Saskatchewan? The choice is yours to make!

Lucky contestant, come on down!	136
"if" this is true... then do something	138
An if statement evaluates a condition... and then takes action	139
Use if to choose between two things	141
You can make multiple decisions with if	142
Adding an else to your if statement	143
Variables drive the story	146
But part of the story is missing	147
Compounding your JavaScript efforts	148
Tiered decision making with if/else	154
An if can go inside another if	155
Your functions control your pages	157
Pseudocode lets you map out your adventure	158
Stick figure inequality	162
!= Psst, I've got nothing to tell you...	163
Crafting decisions with comparison operators	164
Comments, placeholders, and documentation	166
Comments in JavaScript start with //	167
Scope and context: Where data lives	169
Check your adventure variable score	170
Where does my data live?	171
Choice of five	174
Nesting if/else can get complicated	175
Switch statements have multiple cases	177
Inside the switch statement	178
A switchy stick figure adventure: test drive	183



looping

5

At the Risk of Repeating Myself

Some say repetition is the spice of life. Sure, doing something new and interesting is certainly exciting, but it's the little repetitive things that really make it possible to get through the day. Compulsive hand sanitizing, a nervous tick, clicking Reply To All to every freaking message you receive! Okay, maybe repetition isn't always such a great thing in the real world. However, it can be extremely handy in the world of JavaScript.

You'd be surprised how often you need a script to **run a piece of code several times**.

Without **loops**, you'd be wasting a lot of time cutting and pasting a bunch of wasteful code.

Available



seat_avail.png

Unavailable



seat_unavail.png

Select



seat_select.png

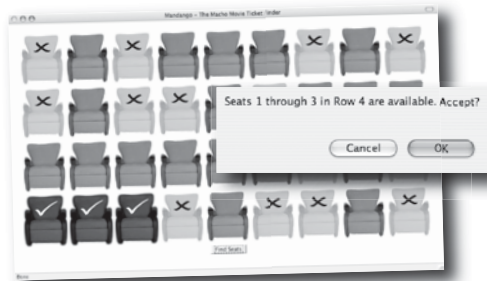
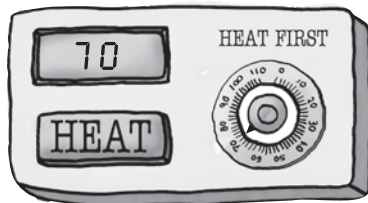
X marks the spot	190
Déjà vu all over again...for loops	191
Treasure hunting with a for loop	192
Dissect the for loop	193
Mandango: a macho movie seat finder	194
First check seat availability	195
Looping, HTML, and seat availability	196
Movie seats as variables	197
Arrays collect multiple pieces of data	198
Array values are stored with keys	199
From JavaScript to HTML	203
Visualizing Mandango seats	204
Test drive: the solo seat finder	209
Too much of a good thing: endless loops	210
Loops always need an exit condition (or two!)	211
A "break" in the action	212
Boolean operator logic uncovered	218
Looping for just a "while"...until a condition is met	222
Breaking down the while loop	223
Use the right loop for the job	225
Movie seat data modeling	231
An array of an array: two-dimensional arrays	232
Two keys to access 2-D array data	233
Mandango in 2-D	235
An entire theater of manly seats	238

functions

6

Reduce, Reuse, Recycle

If there was an environmental movement within JavaScript, it would be led by functions. Functions allow you to make JavaScript code more efficient, and yes, more reusable. Functions are also task-oriented, good at code organization, and excellent problem solvers. Sounds like the makings of a good resume! In reality, all but the simplest of scripts stand to benefit from a functional reorganization. While it's hard to put a number on the carbon footprint of the average function, let's just say they do their part in making scripts as eco-friendly as possible.



The mother of all problems	244
Functions as problem solvers	246
The nuts and bolts of a function	247
A function you've already met	248
Building a better thermostat	251
Passing information to functions	252
Function arguments as data	253
Functions eliminate duplicate code	254
Creating a seat setter function	257
The setSeat() function	259
The significance of feedback	261
Returning data from functions	262
Many happy return values	263
Getting the status of a seat	267
Showing the seat status	268
You can link the function to an image	269
Repetitive code is never a good thing	270
Separating functionality from content	271
Functions are just data	272
Calling or referencing your functions	273
Events, callbacks, and HTML attributes	277
Wiring events using function references	278
Function literals to the rescue	279
Where's the wiring?	280
A shell of an HTML page	283

Forms and Validation

7 Getting the User to Tell All

You don't have to be suave or sneaky to successfully get information from users with JavaScript. But you do have to be careful. Humans have this strange tendency to make mistakes, which means **you can't** always count on the data provided in online forms being **accurate**. Enter JavaScript. By **passing form data through the right JavaScript code** as it is being entered, you can **make web applications much more reliable**, and also **take some load off of the server**. We need to **save that precious bandwidth for important things** like stunt videos and cute pet pictures.

The Bannerocity HTML form	291
When HTML is not enough	292
Accessing form data	293
Form fields follow a chain of events	295
Losing focus with onblur	296
Alert box validation	297
Validate fields to make sure you have “not nothing”	301
Validation without aggravating alert boxes	302
A more subtle non-empty validator	303
Size matters...	305
Validating the length of data	306
Validating a ZIP code	311
Validating a date	316
Regular expressions aren't “regular”	318
Regular expressions define patterns to match	319
Metacharacters represent more than one literal character	321
Drilling into regular expressions: quantifiers	322
Validating data with regular expressions	326
Matching mins and maxes	329
Eliminating three-digit years with this...or that	331
Leave nothing to chance	332
Can you hear me now? Phone number validation	333
You've got mail: validating email	334
The exception is the rule	335
Matching optional characters from a set	336
Constructing an email validator	337

Bannerocity...banner ads in the sky!

wrangling the page

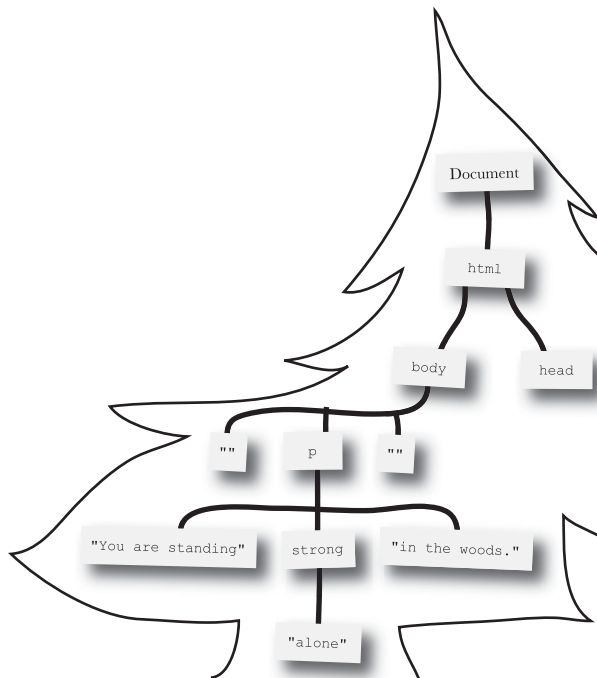
8

Slicing and Dicing HTML with the DOM

Taking control of web page content with JavaScript is a lot like baking. Well, without the mess... and unfortunately, also without the edible

reward afterward. However; you get **full access to the HTML** ingredients that go into a web page, and more importantly, you have the ability to **alter** the recipe of the page. So **JavaScript makes it possible to manipulate the HTML code within a web page** to your heart's desire, which opens up all kinds of interesting opportunities all made possible by a **collection of standard objects** called the **DOM** (Document Object Model).

Functional but clumsy... interface matters	344
Describing scenes without alert boxes	345
Accessing HTML elements	347
Getting in touch with your inner HTML	348
Seeing the forest and the trees: the Document Object Model (DOM)	353
Your page is a collection of DOM nodes	354
Climbing the DOM tree with properties	357
Changing node text with the DOM	360
Standards compliant adventuring	365
Designing better options	367
Rethinking node text replacement	368
Replacing node text with a function	369
Dynamic options are a good thing	370
Interactive options are even better	371
A matter of style: CSS and DOM	372
Swapping style classes	373
Classy options	374
Test drive the stylized adventure options	375
Options gone wrong: the empty button	376
A la carte style tweaking	377
No more bogus options	379
More options, more complexity	380
Tracking the decision tree	382
Building the decision history in HTML	383
Manufacturing HTML code	384
Tracing the adventure story	387



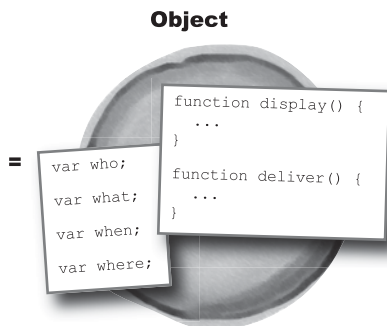
bringing data to life

Objects as Frankendata

9

JavaScript objects aren't nearly as gruesome as the good doctor might have you think. But they are interesting in that they combine pieces and parts of the JavaScript language together so that they're more powerful together. **Objects combine data with actions** to create a new data type that is much more "alive" than data you've seen thus far. You end up with **arrays that can sort themselves**, **strings that can search themselves**, and scripts that can grow fur and howl at the moon! OK, maybe not that last one but you get the idea...

Data	Actions
<pre>var who; var what; var when; var where;</pre>	<pre>function display(what, when, where) { ... } function deliver(who) { ... }</pre>
+	



A JavaScript-powered party	394
Data + actions = object	395
An object owns its data	396
Object member references with a dot	396
Custom objects extend JavaScript	401
Construct your custom objects	402
What's in a constructor?	403
Bringing blog objects to life	404
The need for sorting	409
A JavaScript object for dating	410
Calculating time	411
Rethinking blog dates	412
An object within an object	413
Converting objects to text	416
Accessing pieces and parts of a date	417
Arrays as objects	420
Custom sorting an array	421
Sorting made simple with function literals	422
Searching the blog array	425
Searching within strings: indexOf()	427
Searching the blog array	428
Searching works now, too!	431
The Math object is an organizational object	434
Generate random numbers with Math.random	436
Turn a function into a method	441
Unveiling the shiny new blog object	442
What do objects really offer YouCube?	443

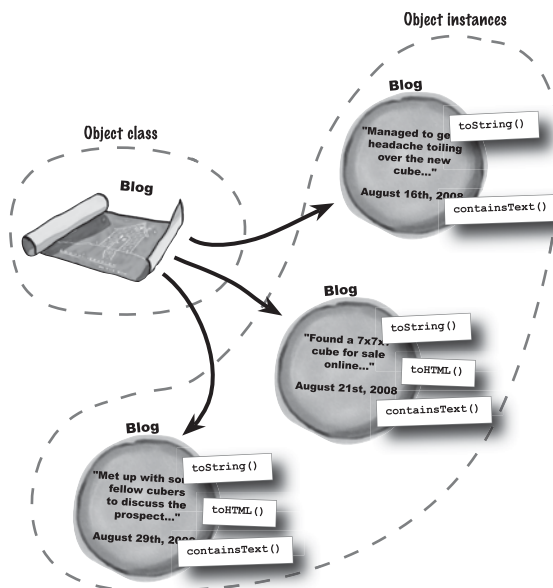
creating custom objects

10

Having It Your Way with Custom Objects

If it was only that easy, we'd surely have it made. JavaScript doesn't have a money-back guarantee, but you can definitely have it your way. Custom objects are the JavaScript equivalent of a decaf triple shot grande extra hot no whip extra drizzle no foam marble mocha macchiato. That is one custom cup of coffee! And with custom JavaScript objects, you can brew up some code that does exactly what you want, while taking advantage of the benefits of properties and methods. The end result is reusable code that effectively extends the JavaScript language...just for you!

Revisiting the YouCube Blog methods	450
Method overload	451
Classes vs. instances	452
Instances are created from classes	453
Access an instance's properties with "this"	454
Own once, run many: class-owned methods	455
Use prototype to work at a class-level	456
Classes, prototypes, and YouCube	457
Class properties are shared, too	462
Creating class properties	463
Signed and delivered	465
A date formatting method	468
Extending standard objects	469
Custom date object = better YouCube	470
A class can have its own method	471
Examine the sort comparison function	473
Calling a class method	474
A picture is worth a thousand words	475
Incorporating images into YouCube	476
Adding imagery to YouCube	478
An object-powered YouCube	480

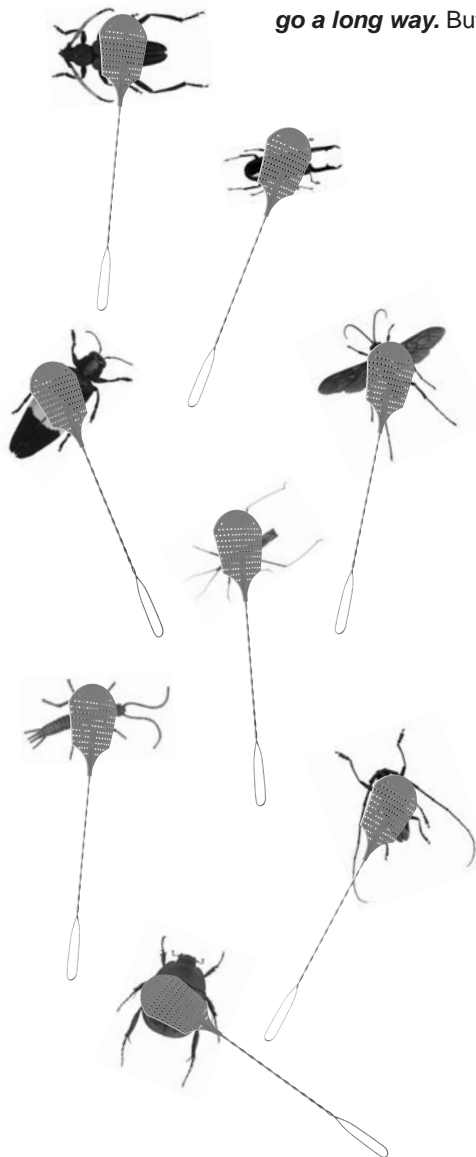


kill bugs dead

Good Scripts Gone Wrong

11

Even the best laid JavaScript plans sometimes fail. When this happens, and it will, your job is not to panic. The best JavaScript developers are not the ones who never create bugs - those people are called liars. No, the best JavaScript developers are those who are able to **successfully hunt down and eradicate** the bugs they create. More importantly, top notch JavaScript bug exterminators **develop good coding habits** that minimize the sneakiest and nastiest of bugs. **A little prevention can go a long way.** But bugs happen, and you'll need an arsenal of weapons to combat them...



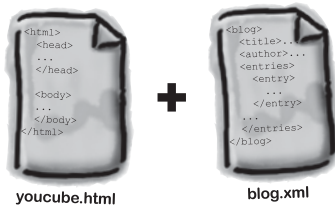
Real-world debugging	486
The case of the buggy IQ calculator	487
Try different browsers	488
Debugging on easy street	491
Variables gone wild undefined	495
Crunching the intelligence numbers	497
The case of the radio call-in bugs	498
Opening up the investigation	499
A question of syntax validation (Bug #1)	500
Careful with those strings	501
Quotes, apostrophes, and consistency	502
When a quote isn't a quote, use escape characters	503
Undefined isn't just for variables (Bug #2)	504
Everyone's a winner (Bug #3)	506
Alert box debugging	507
Watching variables with alert	508
Bad logic is legal but buggy	510
Everyone's a loser! (Bug #4)	514
Overwhelmed by annoying alerts	515
Build a custom console for debugging	517
The peskiest errors of all: runtime	524
The JavaScript bug trifecta	525
Comments as temporary code disablers	528
The dangers of shadowy variables	530

12

dynamic data

Touchy-Feely Web Applications

The modern Web is a very responsive place where pages are expected to react to the user's every whim. Or at least that's the dream of many web users and developers. JavaScript plays a vital role in this dream through a programming technique known as **Ajax** that provides a mechanism for dramatically changing the "feel" of web pages. With Ajax, web pages act much more like full-blown applications since they are able to **quickly load and save data dynamically** while **responding to the user in real time** without any page refreshes or browser trickery.



youcube.html

blog.xml

=



Yearning for dynamic data	538
A data-driven YouCube	539
Ajax is all about communication	541
XML lets you tag YOUR data YOUR way	543
XML + HTML = XHTML	545
XML and the YouCube blog data	547
Injecting YouCube with Ajax	550
JavaScript to the Ajax rescue: XMLHttpRequest	552
Get or Post? A request with XMLHttpRequest	555
Making sense of an Ajax request	559
Interactive pages start with a request object	563
Call me when you're done	564
Handling a response...seamlessly	565
The DOM to the rescue	566
YouCube is driven by its data	571
Dysfunctional buttons	573
The buttons need data	574
Time-saving web-based blog additions	577
Writing blog data	578
PHP has needs, too	581
Feeding data to the PHP script	582
Getting it up: Posting blog data to the server	585
Making YouCube more, uh, usable	590
Auto-fill fields for your users	591
Repetitive task? How about a function?	592