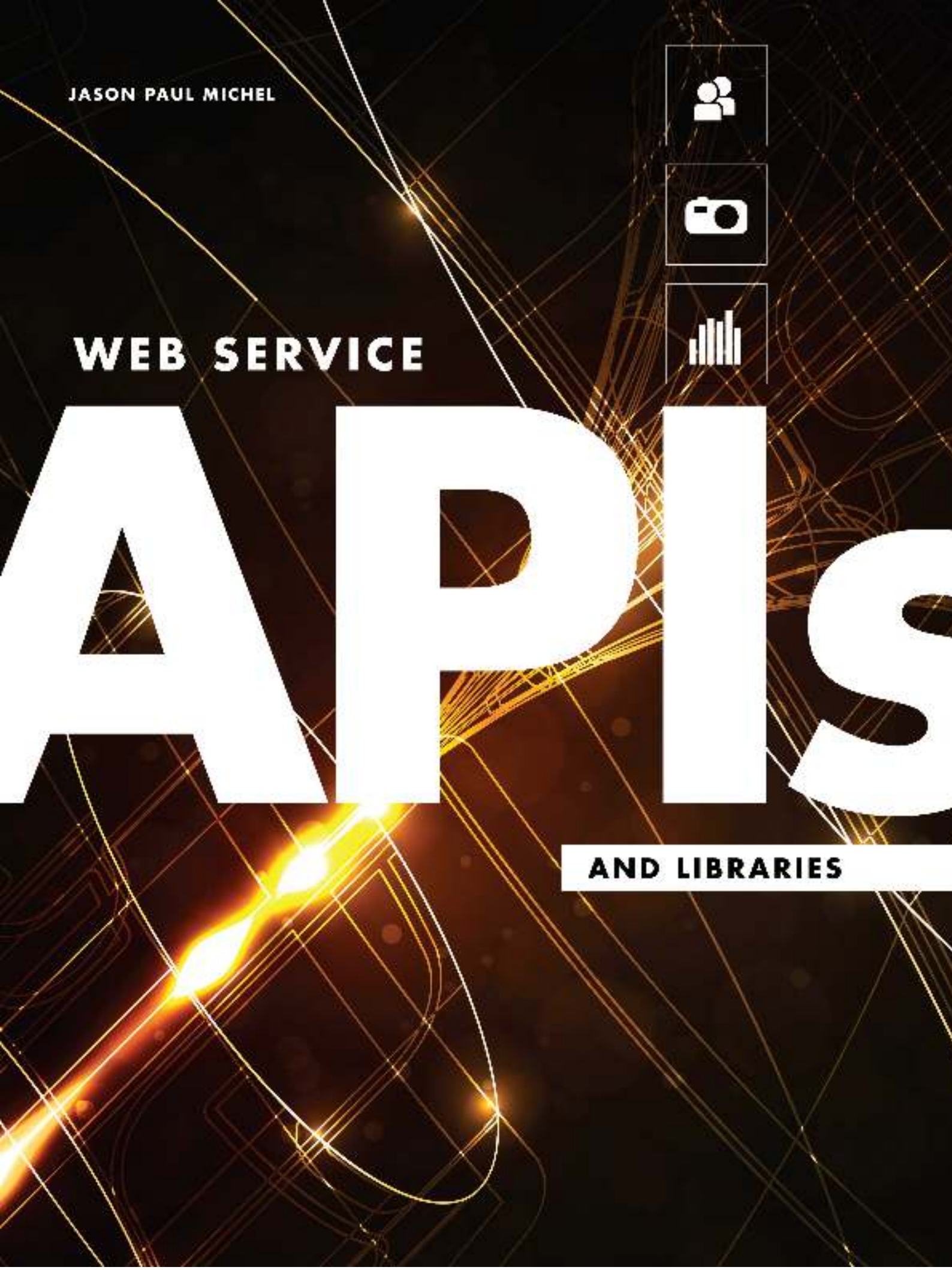

JASON PAUL MICHEL
WEB SERVICE
APIs
AND LIBRARIES

WEB SERVICE

APIs

AND LIBRARIES

WEB SERVICE APIs

AND LIBRARIES

JASON PAUL MICHEL

ala editions

An imprint of the American Library Association
Chicago 2013

Jason Paul Michel is user experience librarian at the Miami University Libraries in Oxford, Ohio. He has engineered numerous API projects at the Miami University Libraries, including Vimeo, Flickr, and Twitter API projects, and has presented on these projects at state and national library conferences.

Printed in the United States of America
17 16 15 14 13 5 4 3 2 1

ISBNs: 978-0-8389-1182-2 (paper); 978-0-8389-9641-6 (PDF). For more information on digital formats, visit the ALA Store at alastore.ala.org and select eEditions.

Library of Congress Cataloging-in-Publication Data
Michel, Jason Paul.
Web service APIs and libraries / Jason Paul Michel.
pages cm.
Includes index.
ISBN 978-0-8389-1182-2 (alk. paper)
1. Web services—Library applications. 2. Application program interfaces (Computer software)
3. Library websites—Design. I. Title.
Z674.75.W67M53 2013
025.042'2—dc23

2012019725

Cover design by Kim Thornton. Composition in Alegreya and Gotham by Casey Bayer.
Cover image © majcot/Shutterstock, Inc.

♾ This paper meets the requirements of ANSI/NISO Z39.48-1992 (Permanence of Paper).

CONTENTS

ACKNOWLEDGMENTS

First, I would like to wholeheartedly thank Christopher Rhodes, editor at ALA Editions. His guidance throughout the work was invaluable. Additionally, I'd like to thank my copy editor, Russell Harper, who transformed my haphazard text into effective prose.

The inspiration for this book came from the projects I pursued at the Miami University Libraries. I must thank Elias Tzoc, with whom I worked on a Flickr API project, the success of which led to further API explorations. I also must thank Rob Casson, whose unparalleled programming expertise is matched only by his willingness to lend a hand.

This book would never have been written if not for the encouraging environment at the Miami University Libraries, as led by Dean Judith Sessions. I also would like to thank the gentle prodding of Assistant Deans Lisa Santucci and Aaron Shrimplin.

Finally, I'd like to thank my wonderful wife, Kristi LaFary. She's the inspiration for everything I do and without whom this book would never have been completed, since as Robert Frost wrote, we are "together wing to wing and oar to oar."

INTRODUCTION

Libraries have lost the battle. The web is now, and has been for a while, where people go to find information.

But that is a vague statement. Imagine the web as a huge, vast cityscape. Most of this city consists of sparsely populated back alleys and warrens, while standing at the center are a few heavily populated monoliths: Google, Wikipedia, Facebook, Twitter, IMDb, Flickr, and a few others. These are the places people go to find information.

Our library websites, including our unique digital collections, are found in one of those back alleys, twenty blocks from Google (twenty pages deep). Unfindable. Dusty. Lonely. All is not lost, however. There is a way to put library websites front and center, and that is through the use of APIs. APIs allow programmers and developers to both deliver content to and receive content from the web's larger, more highly used services. At the same time, APIs give developers the tools and data to create valuable, user-friendly services of their own.

So, what type of library services can be developed through the use of APIs? Let's consider some potential applications.

Twitter

The Twitter API can be utilized to programmatically mine tweets coming from users in your area. You can query this data to see if people are tweeting about your institution, or tweeting about reading, books, doing research, and the like. This insight can give you an opportunity for proactive reference or outreach services.

You could also use the Twitter API as an in-house outage reporting tool. This will be discussed in full detail in the chapter on the Twitter API.

Flickr

The Flickr API can be used in many ways. Imagine you have a large collection of interesting digital images that are hosted on your library's website. The audience for these images is limited if they remain only on your site. Using the Flickr API, you can upload large numbers of digital images and accompanying data in a short amount of time. Sharing these images on Flickr will expose your content to a much wider audience.

You can also use the Flickr API to pull images onto your site. For example, you might use it to serve up Flickr content alongside library content related to a current event. Imagine a web page with a dynamic list of library books and articles related to the 2012 presidential election with a slideshow near the top of the page of Flickr images from the campaign trail.

Vimeo

Sometimes you can use a web service API as a means to deliver your own content to your own site, rather than building up unnecessary infrastructure. Vimeo is a prime example of that. Let's say you have instructional videos for your library. Rather than worrying about how to upload a video to your own server, it may be easier to upload your video content to a site like Vimeo and display those videos on your site using their API. We will go through this process in the chapter on Vimeo's API.

Google Charts

Using the Google Charts API, you can add data visualization aspects to your site. The API could be utilized to display in a graphical way how many computers are available in your computer lab, how many laptops or other pieces of circulating equipment are available for checkout, circulation stats, and more.

Bibliographic Service APIs

Utilizing the APIs of web services such as OCLC, LibraryThing, Open Library, Google Books, and others allows you to develop services that can enrich your library catalog: book covers, book reviews, recommendations, full-text e-books, and more.

I. INTENDED AUDIENCE

This book is intended first of all for library school students. Anyone studying library science today must plan on coming away with a good understanding of web technologies to have any hope of shaping our profession and keeping pace with technology. We, as a profession, will be left in the dust if younger librarians don't understand and embrace these technologies.

It is also intended for librarians and library administrators interested in learning how to better integrate their services into the wider web ecosystem. This, I believe, is a critical mission if libraries are to maintain their relevance.

With respect to technical knowledge, the examples in this book are geared toward people with rudimentary knowledge of HTML, PHP, and HTTP. Complete beginners will be able to learn as they go, however, as long as they follow the examples to the letter.

II. WHAT IS AN API?

An API, or application programming interface, is a set of methods to access data in otherwise closed systems. APIs give programmers and developers the tools necessary to build software and services with data and services from external sources. Extending a metaphor used by David Orenstein,[1] let's imagine that you are building a deck in your backyard and you realize that you don't have a hammer. You have three neighbors who you know have a hammer. One neighbor never allows anyone to borrow anything or use any of his stuff without paying up. This neighbor represents a closed or proprietary system. Another neighbor leaves his garage open and allows you to take anything you need without any rules or guidelines. This is your bearded open-source neighbor. The third neighbor represents an API. You can utilize the services of her hammer as long as you ask her in the proper way. Furthermore, she doesn't actually give it to you; she just allows you to use it.

III. WHAT IS A WEB SERVICE API?

A web service API is an API designed for specific web applications. Web service APIs are one of the hallmarks of the current web ecosystem: an ecosystem based on openness and sharing. Post to Facebook, tweet this, Tumblr that, and so on. The web services we use are no longer closed, autonomous systems. They work together, data flowing freely between them. This is achieved through the use of web service APIs.

IV. TECHNICAL SETUP

Before you will be able to follow along with the examples in this book, you will need to be able to write and post files to the web. For that you will need the following:

1. A web server to which you have write access. Contact your system administrator to get set up with web server space.

2. PHP scripting language, which must be installed on your web server. There is a very good chance that it already is. Contact your system administrator to make sure.
3. A text editor. For beginners I would recommend TextWrangler for Mac or Notepad++ for Windows. For more advanced coders, I would recommend using VI or VIM in the Terminal (Mac) with shell access.
4. An FTP/SFTP client. Many text editors such as TextWrangler have a built-in FTP/SFTP client that allows you to edit your scripts and seamlessly save them to the web server.

The best way to get set up is to speak with your systems folks. They'll be able to get you started with your web development environment.

V. TECHNICAL PRIMER

If you have a solid basic understanding of HTML and PHP, then you can skip this section. Beginners, however, should read through this section. I will introduce the basic HTML and PHP concepts that will be employed throughout the examples in this book.

HTML

HTML is the language of the web. It is used to mark up the *content* of web pages. Most of the scripts written in this book are PHP driven, but we will be using some HTML elements. Therefore, we should introduce some basic concepts of HTML.

HTML Tags

HTML markup is built around the use of HTML tags. Let's start with the basics.

The DOCTYPE declaration is the first tag in an HTML document. It informs the browser which HTML version the web page will be employing:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3
       .org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Next comes the `<html>` tag. All HTML files must include this tag at the top of the document, usually beneath the DOCTYPE declaration. The closing `</html>` tag will be the last tag in the document.

The head section of a web page is enclosed in `<head></head>` tags. This section usually contains the `<title>` of the web page, links to external files such as CSS or JavaScript files, and other metadata.

The body of the HTML document follows the head. Most of the elements within the `<body></body>` tags will be visible on the browser page.

Sections within the body of a web page are denoted by heading tags. The largest heading tag is `<h1>`, so any text between `<h1>` and `</h1>` will be larger than the rest of the text on the page. You can use smaller heading tags by changing the number after *h*: a slightly smaller heading tag would be `<h2>` and so forth—up to `<h6>`.

Paragraphs in an HTML document are enclosed within `<p></p>` tags. When we extract data via APIs later in the book, we will make that data visible within HTML paragraphs and will thus be using this tag quite a bit.

Basic lists are indicated with `<ul></ul>` tags. Much of the data that we will be working with will be in the form of lists—of book titles, tweets, and so forth—and we will be displaying that data in list form on our web pages. Each item in a list is further enclosed within `<li></li>` tags.

Line breaks are indicated by a single `<br />` tag. This tag can also be used to place empty lines within an HTML document for spacing purposes.

You've probably noticed by now that nearly all HTML elements include both opening and closing tags. Thus if you start a paragraph with an opening `<p>` tag, you must end it—after you've put in your text—with a closing `</p>` tag. Note, too, that HTML tags should be lowercase at all times: `<h1>`, not `<H1>`.

Sample HTML Document

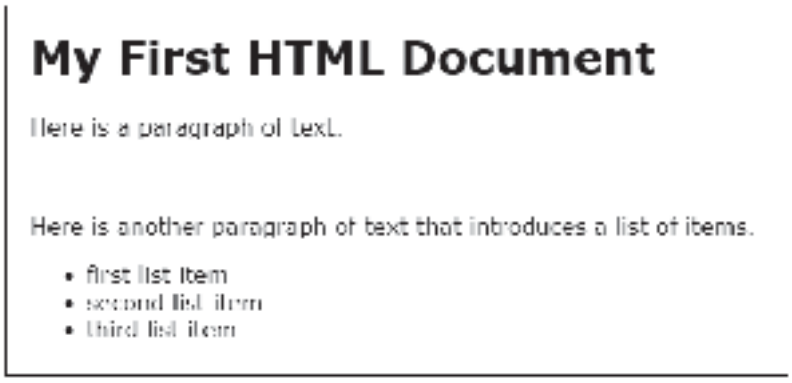
After you have set up your web development environment with your systems administrator, you might want to create a basic HTML document and upload it to your web server space to ensure that everything is set up properly.

Open up your text editor and create a file called `hello_world` and save it as an HTML document before proceeding.

Type the following in to your text editor:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3
       .org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>Hello World!!</title>
</head>
<body>
<h1>My First HTML Document</h1>
<p>Here is a paragraph of text.</p>
<br />
<p>Here is another paragraph of text that introduces a list of items.</p>
<ul>
<li>first list item</li>
<li>second list item</li>
<li>third list item</li>
</ul>
</body>
</html>
```

Now, open up a browser and go to this file on your web server. You should see an HTML page that looks like this:



Now that we have the web environment set up and you understand the basics of HTML, let's move on to the core concepts of PHP that will be employed in this book.

PHP Concepts

PHP, or Hypertext Preprocessor, is a widely used general-purpose scripting language that can be embedded within HTML. All of the files that we create will be PHP files, so you must have PHP installed on your web server for these scripts to work properly.

PHP files begin with a simple PHP declaration:

```
<?php
```

and end with a closing tag:

```
?>
```

Print

The PHP `print` function will print to a web page whatever is contained within quotation marks.

Let's create our first PHP file. Open up your text editor and create a file called `hello_world.php` and save it as a PHP file. Enter the following text into your file:

```
<?php
print "Hello, World!";
?>
```

Notice that the `print` line ends with a semicolon. Nearly all lines of code in a PHP script end with a semicolon. Exceptions include PHP logic such as `if` and `for` statements. We will get into this later.

Variables

Variables are used to store data in a convenient way to use and reuse throughout a PHP document. We will use PHP variables quite a bit in the examples in this book. Declaring a static variable is very simple. Variables always start with `$` and should contain only lowercase text.

Let's declare a simple variable. Open up the PHP file you just created and delete the following line:

```
print '<h1>Hello World!</h1>';
```

Replace it with this:

```
$variable = "This is a variable.";
```

The variable entitled `$variable` can now be used throughout the document. To see how this works let's do a couple of things to the variable. First let's just print it out. We will be printing out variables throughout this book:

```
print '<h1>' . $variable . '</h1>';
```

Save the PHP file and open it in your browser. You should see plain text that says: This is a variable.

You can achieve the same effect by using the echo command:

```
echo $variable;
```

We will employ `echo` a lot in this book to ensure we have our variables correct.

Commenting Out Code

While we work on our scripts, it will sometimes be necessary to disable certain lines of code. This is a common coding need. This is done by *commenting out* code. Let's say that while we are working on a script, we are using the `echo` command quite a bit. This is useful for debugging purposes, but for production we don't want some of those variables `echo`'d out to the browser.

To comment out code we can do one of two things. We can comment out a single line of code by preceding the line with two forward slashes:

```
//echo $variable;
```

PHP will see the two slashes and not perform any PHP commands on that line. If your code spans more than one line, you can use the forward slash and asterisk technique:

```
/* print '<pre>';
print_r ($variable);
print '</pre>';
*/
```

This will comment out all of the content between the opening `/*` and ending `*/`. This can be done for any amount of code that's more than one line.

Arrays

Much of the data that we work with is data stored in PHP arrays. We will not be building arrays, but we will be working with them.

PHP arrays are ordered maps that associate values to keys. To get a basic idea of what an array is, let's make a simple array to work with:

```
$array = array(1,2,3,4,5);
print_r ($array);
```

The command used to print out arrays is `print_r`. Save the file and open it in your browser. The array should be seen as follows:

```
Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 [4] => 5 )
```

The number within brackets is the key, while the number following the `=>` is the value. We will be working with arrays quite a bit in this book.

Logic

PHP allows us to use logic on our variables to make decisions. The two primary logic functions that we will use are `if` and `for`.

`if` Logic

Let's create a new variable to use for an example of how `if` is employed:

```
$var = "1";
```

So `$var` equals one. Let's use `if`:

```
if $var == "1" {
echo $var;
}
```

After our `if` command is the variable that we are checking—in this case `$var`. We are checking to see if `$var` equals one, so we use the `==` operator, which means "is equal to." In quotes is the value that we are checking for. Next we have an opening curly bracket. Notice that we do not have a closing semicolon after the curly bracket. Anything contained in the curly brackets will be completed if the variable equals one. In this case we are just echoing out the variable. Every `if` command must end with a closing curly bracket.

We will be using only the `==` operator in the examples throughout the book. For a complete list of PHP comparison operators see the PHP Manual (at http://php.net/manual/en/language.operators.comparison.php).

`for` Logic

We can use `for` logic to loop through and print out the values of arrays, something we will be dealing with quite a bit in this book.

Let's take another look at the array we created earlier:

```
$array = array(1,2,3,4,5);
```

In order to use `for` logic, we need to know how many values are in the arrays that we are working with. To do that we can employ the `count` command:

```
$count = count($array)
```

What we are doing here is storing the number of values in the `$array` array in another array called `$count`. To ensure that this is working properly, echo out the `$count`:

```
echo $count;
```

Save your PHP file and open it in the browser. You should see a number. It should be 5, if you created the array as mentioned earlier.

Now we can use that variable in a `for` logic loop:

```
for ($i = 0; $i < $count; $i++) {
echo $array[$i];
}
```

When you save this and open it in the browser, you should see the values of the array printed out in order: 12345.

If you wanted to put a line break between the values, you could change what happens within the `for` loop:

```
for ($i = 0; $i < $count; $i++) {
```

```
echo $array[$i];
echo '<br />';
}
```

Let's break the `for` loop down. First we have the `for` command followed by a variable and some commands in parentheses. In the parentheses we are declaring a variable `$i` and assigning it an initial value of 0. Then, after a semicolon, we are saying that while `$i` is less than the value of the `$count` we obtained earlier (i.e., 5), then do what is in the curly brackets. Lastly, we are telling the `for` loop to add 1 to `$i` after each time through the loop. So after it goes through the loop once, `$i` will change its value from 0 to 1, then from 1 to 2, and so forth, until it is no longer less than 5. Once that happens the `for` loop will stop.

This allows PHP to work through the array and for each value in the array to perform the commands in the brackets. Thus for each value in the array it is printing it out and following it with a line break `<br />`.

Data Types

The information that is sent to us via the varied APIs in this book comes in the form of data structures. The two primary data types that we will be working with are JSON (JavaScript Object Notation) and XML. JSON is simpler to work with using PHP and requires only a few built-in PHP functions to parse into PHP arrays. XML data requires some extra massaging of the data to get to a point where we can use it.

HTTP Request

All of the APIs that we are working with are based on HTTP requests. You are familiar with HTTP because you use it every time you visit a web page. The data that we are requesting from the various web services is sent to us via HTTP. You will see that one of the first things we do in each script that we write is to create a URL variable for that web service.

The Elements of a Typical Web Service API

The two primary elements that are inherent to all web service APIs are data and methods. There is considerable variation among the different APIs as to the specifics of these elements, but all APIs will involve data and methods. In addition, some APIs require API keys or authentication, or both. We will take a look at those elements in the individual API chapters. Let's first take a look at the data and the formats you will encounter.

API Data

The format of the data that is provided through most APIs is typically XML (REST, SOAP), JSON, RSS, Atom, and serialized PHP. Web service APIs that provide more than one format option facilitate greater developer flexibility. Here is an example of XML data returned through the Twitter API:

```
<statuses type="array">
 <status>
    <created_at>Wed May 04 23:39:19 +0000 2011</created_at>
    <id>65923490921984000</id>
    <text>Let's get Jen to 200 tweets! Yahoo! http://t.co/qUBlS2d</text>
    <source>
       <a href="http://twitter.com/tweetbutton" rel="nofollow">Tweet Button</a>
    </source>
    <truncated>false</truncated>
    <favorited>false</favorited>
    <in_reply_to_status_id/>
    <in_reply_to_user_id/>
    <in_reply_to_screen_name/>
    <retweet_count>0</retweet_count>
    <retweeted>false</retweeted>
    <user>
       <id>292398272</id>
       <name>Brad Klassen</name>
       <screen_name>logo_man</screen_name>
       <location/>
       <description/>
       <profile_image_url>
        http://a1.twimg.com/profile_images/1337753208/009_normal.JPG
       </profile_image_url>
       <url/>
       <protected>false</protected>
       <followers_count>1</followers_count>
       <profile_background_color>C0DEED</profile_background_color>
       <profile_text_color>333333</profile_text_color>
       <profile_link_color>0084B4</profile_link_color>
       <profile_sidebar_fill_color>DDEEF6</profile_sidebar_fill_color>
       <profile_sidebar_border_color>C0DEED</profile_sidebar_border_color>
       <friends_count>3</friends_count>
       <created_at>Tue May 03 15:51:05 +0000 2011</created_at>
    </user>
 </status>
</statuses>
```

SANDBOX EXAMPLE 1

Let's access Twitter's public status time line. Enter the API request URL into your browser:

```
http://api.twitter.com/1/statuses/public_timeline.format
```

Now replace `format` with `xml`. You should see data similar to the XML data in the example above. Now try entering `rss` or `atom`. The data you see now should look a bit more familiar to you, especially if you are used to subscribing to RSS feeds. Most APIs are flexible enough to give you multiple format options so that, depending on your project, you can choose the one that works best.

API Methods

How do developers manipulate data via an API? Primarily through simple HTTP GET and POST requests. For example, the data above was retrieved via an HTTP GET request using the following URL template:

```
http://api.twitter.com/1/statuses/public_timeline.format
```

Each API has several methods to access different types of data. For example, the above method retrieved the public time line of statuses of all Twitter users. Other methods allow you to retrieve other data such as Twitter trends.[2] The individual chapters in this book will explicate these methods in detail.

VI. WHY ARE APIs IMPORTANT FOR LIBRARIES?

APIs are important for libraries because they allow them to connect with the wider web ecosystem and to integrate with prominent web services such as Twitter, Flickr, Facebook, and more. The current social media environment is one of interconnectedness. APIs allow libraries to join in that interconnectedness.

In addition to this overarching benefit, APIs have the ability to change library workflows and enhance existing services. For example, working with the Twitter API, libraries can automatically tweet to a specific Twitter user when a study room becomes available, or set up a web-based dashboard that allows users and staff to see if there are any computer or printer problems (see chapter 1).

VII. CHAPTER STRUCTURE

For ease of use, each chapter is structured in the same manner. Each begins with a discussion of services that could be provided through the use of the respective API. This section is followed by a thorough explanation of the API. Finally, each chapter features an "API in Action" section that details the construction of an application in a step-by-step manner. (One exception: chapter 3 is organized around a real-life case study involving the Vimeo API.)

For those of you who like lists, here's another look at what each chapter offers:

- Potential services achieved through use of the API
- Real-life examples of how the API is being used in libraries
- Technical explanation of the API
- API in Action: a step-by-step construction of the application

Note that most of the applications we create together will lack the styling and presentation that comes with further web development using CSS and HTML. Instead of focusing on aesthetic presentation with our scripts, we will be learning how to gather, parse, and deliver data.

VIII. ADDITIONAL RESOURCES

I will be guiding you step-by-step through each script, so you can reasonably work through them without consulting other resources. However, you may want to refer to other resources to clarify certain functions or to answer any questions you may have. I list here some of the better places to go for such information.

HTML

HTML Tutorial, www.w3schools.com/html/.

W3Schools.com is a very good resource to help one learn the basics of many web technologies, not just HTML. Each technology is presented through a series of step-by-step tutorials with a "Try it Yourself" editor that allows you to write code and see the results in your browser without the need to set up a web server. In addition to these two features, W3Schools.com also has an extensive reference section and glossary.

PHP

- PHP Tutorial, www.w3schools.com/php/.
- PHP Manual, www.php.net/manual/en/.
  The PHP Manual offers complete explanations of all of the functions and operators we use for the scripts that we construct throughout this book. This is the official site for PHP.

- Welling, Luke, and Laura Thomson. *PHP and MySQL Web Development*. 4th ed. Upper Saddle River, NJ: Addison-Wesley, 2008.
  A very good introduction to the concept of database-driven dynamic coding with PHP, this book complements many of the concepts and scripts we explore throughout this book.

Twitter API

- https://dev.twitter.com/.
  This is the official website for the Twitter development community. Any bugs or changes to the API will be reported upon and discussed here. Each API function is delineated in full detail. This resource should be continually open in your browser as you work through the Twitter API section.
- Makice, Kevin. *Twitter API: Up and Running*. Sebastopol, CA: O'Reilly Media, 2009. http://shop.oreilly.com/product/9780596154622.do.
  Another great introductory work specifically for the Twitter API. Though Makice's examples are not specific to libraries, his work explains in great detail the core concepts we cover in this book.

Flickr API

- www.flickr.com/services/api/.
  This is the official website for Flickr's API. Along with technical documents on all aspects of the API, there is a great Flickr developer community that can be tapped into through this resource.
- Richardson, Leonard, and Sam Ruby. *RESTful Web Services*. Sebastopol, CA: O'Reilly, 2007. http://shop.oreilly.com/product/9780596529260.do.

Vimeo API

https://developer.vimeo.com/.
The official web resource for Vimeo's API and development platform. The code that we write throughout the Vimeo chapter is well documented on this site.

Google Charts API

https://developers.google.com/chart/.

OCLC Web Services

http://oclc.org/developer/webservices.

HathiTrust API

www.hathitrust.org/data_api.

Open Library API
http://openlibrary.org/developers/api.

LibraryThing API
www.librarything.com/api.

Goodreads API
www.goodreads.com/api.

Google Books API
https://developers.google.com/books/docs/v1/using.

NOTES

1. David Orenstein, "QuickStudy: Application Programming Interface (API)," *Computerworld*, January 10, 2000, www.computerworld.com/s/article/43487/Application_Programming_Interface.
2. "GET trends/:woeid," https://dev.twitter.com/docs/api/1/get/trends/%3Awoeid.