Real World Web Services

It’s no secret that web services continue to be one of the most hyped, most touted, and most popular developments in network computing over the last five years. However, with hundreds of specifications emerging, it’s still unclear—and almost impossible—to turn REST, RDF, SOAP, XML, and the rest into something that’s actually useful. Finally, though, there is a clear, concise guidebook for Java developers seeking to leverage the power of web services, today: Real World Web Services.

Real World Web Services dispenses with the idea that just because something is a specification, it must be used. Instead, a practical (and often critical) eye is applied to the problems facing Java developers. This book isn’t about theory; instead, it focuses on using working web service APIs, from important industry players such as:

- PayPal
- Amazon.com
- Google.com
- CDDB (the database that powers Windows Media Player)
- eBay
- FedEx

If you’re a Java developer who wants to write code that works—and works today—this is the book for you. With eight extensive projects combining popular web service APIs, and code on every page, this book is for the Java journeyman. So toss out the 100-page specifications, and start doing real work, with Real World Web Services.

Will Iverson has been working in the computer and information technology field professionally since 1990. His diverse background includes developing statistical applications to analyze data from the NASA Space Shuttle, product management for Apple Computer, and developer relations for Symantec’s VisualCafé. For nearly five years, Will ran an independent J2EE consulting company with a variety of clients including Sun, BEA, and Canal* Technologies. Will currently serves as the application development practice manager for SolutionsIQ.
Real World Web Services

It’s no secret that web services continue to be one of the most hyped, most touted, and most popular developments in network computing over the last five years. However, with hundreds of specifications emerging, it’s still unclear—and almost impossible—to turn REST, RDF, SOAP, XML, and the rest into something that’s actually useful. Finally, though, there is a clear, concise guidebook for Java developers seeking to leverage the power of web services, today: Real World Web Services.

Real World Web Services dispenses with the idea that just because something is a specification, it must be used. Instead, a practical (and often critical) eye is applied to the problems facing Java developers. This book isn’t about theory; instead, it focuses on using working web service APIs, from important industry players such as:

- PayPal
- Amazon.com
- Google.com
- CDDB (the database that powers Windows Media Player)
- eBay
- FedEx

If you’re a Java developer who wants to write code that works—and works today—this is the book for you. With eight extensive projects combining popular web service APIs, and code on every page, this book is for the Java journeyman. So toss out the 100-page specifications, and start doing real work, with Real World Web Services.

Will Iverson has been working in the computer and information technology field professionally since 1990. His diverse background includes developing statistical applications to analyze data from the NASA Space Shuttle, product management for Apple Computer, and developer relations for Symantec’s VisualCafé. For nearly five years, Will ran an independent J2EE consulting company with a variety of clients including Sun, BEA, and Canal+ Technologies. Will currently serves as the application development practice manager for SolutionsIQ.
Real World Web Services
### Other Java™ resources from O’Reilly

**Related titles**

- Java™ in a Nutshell
- Java™ and XML
- Java Servlet and JSP Cookbook™
- Java™ Web Services in a Nutshell
- Learning Java™
- Head First Java™
- Google Hacks™
- eBay Hacks™
- PayPal Hacks™

**Java Books Resource Center**

djava.oreilly.com is a complete catalog of O’Reilly’s books on Java and related technologies, including sample chapters and code examples.

**OnJava.com** is a one-stop resource for enterprise Java developers, featuring news, code recipes, interviews, weblogs, and more.

**Conferences**

O’Reilly brings diverse innovators together to nurture the ideas that spark revolutionary industries. We specialize in documenting the latest tools and systems, translating the innovator’s knowledge into useful skills for those in the trenches. Visit conferences.oreilly.com for our upcoming events.

**Safari Books Online** (safaribooksonline.com) is the premier online reference library for programmers and IT professionals. Conduct searches across more than 1,000 books. Subscribers can zero in on answers to time-critical questions in a matter of seconds. Read the books on your Bookshelf from cover to cover or simply flip to the page you need. Try it today with a free trial.

---

**easy computing**
Table of Contents

Preface ........................................................................................................ vii

1. Web Service Evolution ................................................................. 1
   Client/Server Origins ................................................................. 2
   The Undefined Web ..................................................................... 3
   Planning for Interdependence ................................................... 7

2. Foundations of Web Services ..................................................... 9
   Basic Networking ...................................................................... 9
   HTTP ...................................................................................... 14
   From HTTP to RPC .................................................................. 17

3. Development Platform ............................................................. 22
   Tools and Projects Used .......................................................... 22
   Test Drive .............................................................................. 25
   Other Platforms ..................................................................... 33

4. Project 1: Competitive Analysis ............................................. 35
   Application Features ................................................................ 35
   Gathering Web Service Data ................................................... 45

5. Project 2: Auctions and Shipping ........................................... 73
   Auction Listing XML .............................................................. 73
   Processing the Auction XML ................................................ 75
   Connecting to FedEx ............................................................. 81
<table>
<thead>
<tr>
<th>Project</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.</td>
<td>Project 3: Billing and Faxing</td>
<td>95</td>
</tr>
<tr>
<td></td>
<td>Starting the Transaction</td>
<td>97</td>
</tr>
<tr>
<td></td>
<td>Getting a Transaction Notification</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td>Responding to the Transaction</td>
<td>107</td>
</tr>
<tr>
<td>7.</td>
<td>Project 4: Syndicated Search</td>
<td>113</td>
</tr>
<tr>
<td></td>
<td>Making Feeds Available</td>
<td>114</td>
</tr>
<tr>
<td></td>
<td>Using an Aggregator</td>
<td>122</td>
</tr>
<tr>
<td>8.</td>
<td>Project 5: News Aggregator</td>
<td>125</td>
</tr>
<tr>
<td></td>
<td>Getting Started</td>
<td>125</td>
</tr>
<tr>
<td></td>
<td>Deleting a Watcher</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>Current Watcher Listing</td>
<td>131</td>
</tr>
<tr>
<td></td>
<td>Watcher Implementation</td>
<td>133</td>
</tr>
<tr>
<td></td>
<td>Going Further with Quartz</td>
<td>154</td>
</tr>
<tr>
<td>9.</td>
<td>Project 6: Audio CD Catalog</td>
<td>155</td>
</tr>
<tr>
<td></td>
<td>CDDB</td>
<td>155</td>
</tr>
<tr>
<td></td>
<td>Building a CD Catalog</td>
<td>156</td>
</tr>
<tr>
<td>10.</td>
<td>Project 7: Hot News Sheet</td>
<td>165</td>
</tr>
<tr>
<td></td>
<td>Presenting the News</td>
<td>165</td>
</tr>
<tr>
<td></td>
<td>Gathering the News</td>
<td>169</td>
</tr>
<tr>
<td>11.</td>
<td>Project 8: Automatic Daily Discussions</td>
<td>176</td>
</tr>
<tr>
<td></td>
<td>Weblogs</td>
<td>176</td>
</tr>
<tr>
<td></td>
<td>Generating Posts</td>
<td>178</td>
</tr>
<tr>
<td></td>
<td>Generating a LiveJournal Post</td>
<td>184</td>
</tr>
<tr>
<td></td>
<td>Viewing the Blogger Post</td>
<td>189</td>
</tr>
<tr>
<td>12.</td>
<td>Future Web Service Directions</td>
<td>192</td>
</tr>
<tr>
<td></td>
<td>Future Technologies</td>
<td>192</td>
</tr>
<tr>
<td></td>
<td>Future Directions</td>
<td>197</td>
</tr>
</tbody>
</table>

Index                                           203
The core idea behind this book is simple: after years of hype, what are the major players really doing with web services? Standard bodies may wrangle and platform vendors may preach, but what technologies are actually in use?

Think of this book as a field guide to the wild and woolly world of nontrivial deployed web services. The heart of the book is a series of projects that demonstrate the use and integration of Google, Amazon, eBay, PayPal, FedEx, and many more web services. Some of these vendors have been extremely successful with their web service deployments; for example, eBay processes over a billion web service requests a month.

Not all web services are created equal: some rely on a variety of strange formats; others require extensive and error-prone XML; and still others require a minimal knowledge of SOAP and WSDL. This book provides compelling examples of the value of SOAP and WSDL for the client developer. In Chapter 4, for example, you can compare the custom bindings required for working with complex XML data types against the generation of SOAP binding generation from WSDL.

**Organization**

This book is divided into four sections: introductory material, a conceptual orientation with regard to web services, the various projects surveying real world web service deployments, and finally, a brief chapter outlining some thoughts on the future of web services. While experienced developers may be inclined to skim the introductory chapters, it may be worth covering them again just to make sure you haven’t missed the forest for the trees.

Chapter 1, *Web Service Evolution*

This chapter provides a high level, business-oriented introduction to web services. Technology serves human needs, and this chapter shows how web services fill an important role in the development of the Web. Many developers
maybe familiar with these concepts already, but it is as important to explain why you do a thing as to explain how it is done.

Chapter 2, Foundations of Web Services
While Chapter 1 covers a business-oriented approach to the history of web services, this chapter provides a technical history. Seasoned developers may find this all familiar, but for readers just joining the web services conversation, this is vital background information. In addition, this chapter can serve as a useful checklist for planning your own web service development and deployment plans.

Chapter 3, Development Platform
This chapter provides an introduction to the Java™ development platform and tools used in the projects in this book. An introduction is given to Apache Tomcat, Apache XML-RPC, and Apache Axis, the web server, XML-RPC, and SOAP/ WSDL toolkits respectively. Obviously, Java is not the only development platform available, and a brief discussion of alternatives concludes the chapter.

Chapter 4, Project 1: Competitive Analysis
The first project in the book, this chapter shows how data from Amazon, eBay, and Google can be used to present an integrated report to a user. Connectivity to each of these three web service providers is shown, providing an example of the developer effort required to access each system.

Chapter 5, Project 2: Auctions and Shipping
This chapter shows how FedEx and eBay can be integrated to provide auction listings with precalculated FedEx shipping estimates. XML is used throughout—from the local auction listings to FedEx and eBay web services.

Chapter 6, Project 3: Billing and Faxing
In this chapter, high-tech web services are used to integrate PayPal billing with low-tech fax technology.

Chapter 7, Project 4: Syndicated Search
This chapter illustrates a web service gateway, using a Google search result to provide a syndicated RSS feed.

Chapter 8, Project 5: News Aggregator
While other examples in this book operate in direct response to user interaction, this chapter uses the Quartz scheduler framework to monitor Amazon, eBay, Google, and RSS feeds on a regular, reliable schedule.

Chapter 9, Project 6: Audio CD Catalog
This chapters shows how CDDB and Amazon can be used together to create a catalog of your audio CDs.

Chapter 10, Project 7: Hot News Sheet
In this chapter, you’ll build an application using RSS to provide a single web page showing what’s hot both from the mainstream news and the weblog universe, side by side. Collaboration will additionally fold in results from a Google search on these topics for yet another angle on the news.
Chapter 11, *Project 8: Automatic Daily Discussions*
In this chapter, you’ll build an application to combine your Blogger or LiveJournal weblog with Google’s search functionality to create automatic prompts for daily discussions.

Chapter 12, *Future Web Service Directions*
This chapter starts with a look at some of the more futuristic web service technologies, including REST, UDDI, Rendezvous, and BPEL/BPEL4WS. A look at the future of web service development follows, which considers ease-of-use, the need for a business model, security, and finally, the consolidation of web services.

**Conventions Used in This Book**

The following font conventions are used in this book:

*Italic* is used for:
- Unix pathnames, filenames, and program names
- Internet addresses, such as domain names and URLs
- New terms where they are defined

*Boldface* is used for:
- Names of GUI items (window names, buttons, menu choices, etc.)

*Constant width* is used for:
- Command lines and options that should be typed verbatim
- Names and keywords in Java programs, including method names, variable names, and class names
- XML element names and tags, attribute names, and other XML constructs that appear as they would within an XML document

Indicates a tip, suggestion, or general note.

Indicates a warning or caution.

**Using Code Examples**

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you’re reproducing a significant portion of the code. For example,
writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product’s documentation does require permission.

We appreciate, but don’t require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: “Real World Web Services, by Will Iverson. Copyright 2004 O'Reilly Media, Inc., 0-596-00642-X.”

If you feel your use of code examples falls outside fair use or the permission given here, feel free to contact us at permissions@oreilly.com.

Comments and Questions

Please address comments and questions concerning this book to the publisher:

O’Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the U.S. or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

There is a web page for this book, which lists errata, examples, and additional information. You can access this page at:

http://www.oreilly.com/catalog/realwws

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about books, conferences, Resource Centers, and the O’Reilly Network, see the O’Reilly web site at:

http://www.oreilly.com

Acknowledgments

First, I must send my gratitude to the unseen engine behind the production, marketing, and sales of this book. I know you’re there and appreciate all your efforts. Thanks to everyone who has contributed in one form or another to the technologies underlying and comprising web services. We so blithely take for granted so much of this innovation—we stand on the shoulders of giants.

This book wouldn’t be what it is without the efforts of my editor, Brett McLaughlin. His work made for a far better book. Thanks to Tim O’Reilly for all the fine tomes
that grace my bookshelf and for allowing me to add a second title to O'Reilly’s excellent library. For their time, enthusiasm, and professional support, I would like to thank Brian Lawley and Daniel Steinberg. Has it really been that long?

On a personal note, thanks to friends and family for their support. And finally, thanks to Mom, Diane, and Cynthia. You are, quite simply, the best.
CHAPTER 1

Web Service Evolution

What are web services? While this might seem a simple question, this book demonstrates that the query has many answers. Much of this is because the typical conversation about web services suffers from the blending of several distinct concepts. Most software developers focus on the technical underpinnings that make communication possible (such as SOAP and XML-RPC). Others add to the web services category developer infrastructure, such as WSDL, the Web Services Description Language. Some even include a wide host of other pieces, including a mind-numbing array of standards (some real, some theoretical).

Because of this confusion, we must define what is meant by the term web services. Here’s a good start:

Web Services: A vague term that refers to distributed or virtual applications or processes that use the Internet to link activities or software components. A travel Web site that takes a reservation from a customer, and then sends a message to a hotel application, accessed via the Web, to determine if a room is available, books it, and tells the customer he or she has a reservation is an example of a Web Services application.

–Business Process Trends
http://www.bptrends.com/resources_glossary.cfm?letterFilter=W&displayMode=all

This is a great start, but it still needs to be clarified a bit. This book doesn’t engage in an intellectual debate as to the “correctness” of web services on a theological level. Instead, it focuses on the practical, real world usage of web services. For this book’s purposes, web services are the latest evolution in distributed computing, allowing for structured communication via Internet protocols. As you’ll see, this includes everything from sending HTTP GET commands to retrieving an XML document through the use of SOAP and various vendor SDKs.
Client/Server Origins

Most developers are familiar with the basic concept of client/server computing: a central server handles requests from one or more clients. It’s the foundation of the Web as described by HTTP and HTML: a web server sends pages to a browser client.

As shown in Figure 1-1, you can see that the client browser initiates the request, which is then processed and responded to by the web server.

![Figure 1-1. Web concept of client server](image)

Before the Web really took off, the term client/server was used to describe something slightly different—a central database server that was accessed by custom clients (often written in Visual Basic, C, Pascal, or other languages). Nobody really wanted to run around, constantly installing and reinstalling these custom clients every time there was a bug to be fixed, so people quickly started hooking up smarter, more sophisticated web servers to databases. In order to distinguish these smarter web servers from “dumb” web servers (that served mainly static files), the term application server became popular (in some circles, you’ll hear the term web container used as well, usually when the server does a lot more than just handle web data).

As you can see in Figure 1-2, the notion of a client and a server becomes more complex in these systems, often called three-tier systems. The browser is clearly acting as a client of the application server, but the application server is a client of the database. In effect, the application server is acting both as a client of the database and a server for the client browser.

Consider the system shown in Figure 1-3: note the number of connections, and the number of different roles (client and server) being played by the different systems. These more complex systems are referred to as n-tier systems, because they can have any number of tiers.

As a mental exercise, consider the following:

- How do you debug this system?
- Each network connection incurs a certain latency and bandwidth overhead. How many connections are appropriate? What happens to the rest of the system if one machine “starves” another by taking all its bandwidth or available connections?
• How are these connections secured?
• What happens if one or more systems crash or otherwise become unavailable?
• How do you roll out new applications?

These are tricky problems. When you’re working with web services, you’ll want to keep these potential pitfalls in mind.

The Undefined Web

The concept of a browser talking to a web server is perhaps the most popular client/server system devised (email is the other one). It didn’t take very long before
the popularity of this model lead to some interesting questions about the proper relationship between the client and the server.

**Scraping Data**

A couple of web sites, desperate for content, realized that they could *scrape* the HTML of other sites and display some or all of that information in a different format. For example, let’s say that you ran a small web site devoted to the glories of Davis, CA. As shown in Figure 1-4, you set up a site that grabs the weather report from another site (steps 2 and 3) and then grabs the stock quote for the public corporation that runs the local gas station (steps 4 and 5). The user can visit your site and get your information as well as the data from the other two sites as well; throw a banner ad at the top of the page, and you’ll soon be rich!

![Diagram showing the process of scraping](image)

*Figure 1-4. An application server scraping other sites*

The problem with scraping (dubious ethics aside) is that HTML is extremely fragile. The only promise given with HTML is that a browser can render properly formatted HTML in a human-readable format, and even that’s a bit of a reach sometimes. A very minor formatting change can break your HTML parser, and the operator of the site doesn’t care (or is actively trying to foil your attempts to steal content).

Now, let’s take this to the next logical step. Let’s say the weather and stock guys notice that you’re reading their data, and both call you and generously offer to trade you legitimate access to their data in exchange for links back to their site. You agree,
and now you need to set this up. The immediate question becomes: what standards and specifications do you use to tie all this information together?

This is perhaps one of the most contentious and controversial aspects of web services. How do you decide the actual implementation details for how these systems are going to talk to each other?

---

**The Dangers of Vendor Lock In**

Most developers who have worked in the field for a while have lumps of code that are, for one reason or another, no longer viable. Common reasons include a shift in the popularity of given programming languages, changes in the market share or architecture of software platforms, or the release of new operating systems. Regardless of the reason, it’s a painful day when you wake up and realize that your 300,000-line, lovingly hand-crafted Pascal code for Mac OS 7 is now essentially worthless, or that you’ll never find anyone to help maintain those 28,000 lines of your favorite assembly program. That fear of platform lock in—you wake up one day and find that your code is now worthless because the vendor that provided the service has gone out of business—is a pretty powerful reason to delay adding web service consumption to your list of features.

As we work our way through the examples in this book, I’ll note possible failure areas for vendors and strategies for dealing with these problems.

---

**Fragile Interdependence**

One of the most significant problems when trying to figure out how to get two systems to talk to each other is sorting out what dependencies, assumptions, and standards to use. For example, we assume that we will be using TCP/IP and the other core technologies of the Internet, but we may not (for example) be comfortable assuming that our partners are willing to standardize on Java or .NET technologies. Instead of declaring required technologies by fiat, our first instinct is to wait and see what standards get locked down.

Preferably, the standards we choose have several solid implementations and have been in use for some time. This allows us to understand more of the pros and cons of any particular technology. HTML, for example, has been in use for some time, but different web browsers can have wildly different interpretations of a given HTML document. Many of the same problems you see with HTML can be seen with web services; for example, consider the seemingly simple questions of style and perspective reflected in the differences between the HTML pages shown in Examples 1-1 and 1-2 (both display the same text on screen).
Example 1-1. Simple HTML

```html
<HTML>
  <HEAD>
  </HEAD>
  <BODY>
    <P ALIGN="CENTER"><B>This is my text!</B></P>
  </BODY>
</HTML>
```

Example 1-2. Complex HTML

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <style type="text/css">-->
      P { font-style: normal; font-weight: bold;
       text-align: center; }
    </style>
  </head>
  <body>
    <p>This is my text!</p>
  </body>
</html>
```

Sometimes differences are merely a matter of style and not substance. For example, consider the differences in method naming standards between Java and Microsoft C/C++. Java developers typically prefer relatively verbose naming, with a strong object-as-noun, method-as-verb nomenclature, heavily influenced by the patterns put forth by the JavaBeans specification that you’ll find at:


Microsoft developers are more likely to use Hungarian notation, which as even Microsoft notes, “make the variable names look a bit as though they’re written in some non-English language”; see the following for more information:


However, .NET is phasing this out; see the following:


While style issues are relevant when you talk about web services—as you’ll see, a perfectly usable set of web service interfaces provided by a vendor can still feel very awkward if the interfaces are based on another style and mental model—the important thing is that services can still be accessed in a reliable, predictable manner. The goal when using web services is to get away from wildly undefined and fragile processes (such as scraping HTML) and instead move toward refined, manageable systems.
Planning for Interdependence

The pundits, purveyors, and snake oil salesmen of web services describe a world in which nearly every process is handled seamlessly by a variety of different web service technologies and options. Automated agents, interconnected by wired and wireless technologies, will use web services to solve global economic crises, find you the best deal on toasters, and restock your refrigerator with fresh milk before you’ve even noticed that the expiration date has passed.

This deeply interconnected world assumes that the underlying web services work very well. In particular, such broad-reaching automation leads to basic questions about responsibilities. For example, let’s say that an error leads to your refrigerator ordering 500 gallons of milk—or none at all. Where did the problem occur? This issue affects you both as a user and provider of web services.

To manage questions of reliability, you must determine what sort of uptime you provide (or demand) for your web services. How do you characterize the performance and security restrictions? What are the implications of a service failure?

It can be helpful to build a brief worksheet when working with web services that can be used as a check list. It can include:

- Number of web service methods exposed or used
- Frequency of access allowed (e.g., one method call per second at most for methods \( a() \), \( b() \), \( c() \), and one call of method \( d() \) per minute)
- Expected performance of specific methods
- Time expected to restore service (e.g., if there is a failure, how long until it’s fixed?)
- Bandwidth and latency expectations
- Scheduled downtime
- Data management and backup responsibility
- Logging
- Security auditing

Here are some questions to ask yourself:

- What are your internal plans for migration if the service fails?
- What is the involvement of your legal representation in drafting or agreeing to your service agreement?
- What tools or systems will you use to monitor your services? (If you promise to deliver or receive a given level of service, how will you really know?)
- What is the expected level of tech support access? How long until a response is sent?
• How will minor bug fixes and upgrades be handled? Will users of the web services be able to test their application against a test server before the changes are pushed to the production system?

• How often will new functionality be added? What is the procedure for migrating web service users to new systems?

• How long will session data be preserved? (In other words, if I begin a transaction, how long does the remote system maintain that state data before expiring it?)

• What are the remedies (refunds or credits) if service fails?

Notice that there is no mention of the programming language used, the application server, the database, the server hardware—all critical to the internal development conversation, but (in theory, at least) not part of the overall web services conversation between two different organizations.

Because of the complexities of interdependence, when you’re working with web services, you need at least a basic understanding of the underlying networking principles, which are discussed in the next chapter.
About the Author

Will Iverson has been working in the computer and information technology field professionally since 1990. His diverse background includes developing statistical applications to analyze data from the NASA Space Shuttle, product management for Apple Computer, and developer relations for Symantec’s VisualCafé. For nearly five years, Will ran an independent J2EE consulting company with a variety of clients including Sun, BEA, and Canal+ Technologies. Will currently serves as the application development practice manager for SolutionsIQ. He lives in Seattle, Washington.

Colophon

Our look is the result of reader comments, our own experimentation, and feedback from distribution channels. Distinctive covers complement our distinctive approach to technical topics, breathing personality and life into potentially dry subjects.

The animal on the cover of Real World Web Services is a domestic pigeon (Columba livia). There are more than 150 breeds of domestic pigeon, in a variety of sizes, colors, and patterns. The typical domestic pigeon is distinguished by its blue and gray plumage. As adults, domestic pigeons are approximately 12 to 14 inches long and can weigh up to 3.5 pounds. Their traditional diet consists primarily of seeds and whole grains, and they can travel far from their nest to locate food. Domestic pigeons reach sexual maturity at five to seven months of age. They mate throughout the year but predominantly in the summer months. The female pigeon lays two eggs each mating cycle, and the male and female birds take turns sitting on the eggs to incubate them until they hatch.

The domestic pigeon is thought to be the first bird tamed by humans. The first domestic pigeons were bred about 6,000 years ago from the rock pigeon, which lived in the wild in Europe, the Middle East, and Southwest Asia. The domestic pigeon was first brought to North America in the early 1600s. Today, American domestic pigeons thrive in urban areas, where they have become comfortable amid the bustle and noise of city life and have adapted their diet to survive on leftover scraps of human food.

One of the most famous breeds of domestic pigeon is the carrier pigeon. Bred for the pigeon’s exceptional homing abilities, the carrier pigeon has been used since ancient times to transmit written messages fastened to its body. When dispatched, the carrier pigeon can travel at a speed of 45 miles per hour, and despite traversing extremely long distances, it instinctively returns to its home coop. During World Wars I and II, carrier pigeons saved hundreds of human lives, intrepidly flying through combat zones to deliver crucial messages at times when radio transmissions weren’t feasible.

Mary Anne Weeks Mayo was the production editor and copyeditor, and Sarah Sherman was the proofreader for Real World Web Services. Sanders Kleinfeld, Emily
Quill, and Claire Cloutier provided quality control. Mary Agner provided production assistance. John Bickelhaupt wrote the index.

Ellie Volckhausen designed the cover of this book, based on a series design by Edie Freedman. The cover image is a 19th-century engraving from the Dover Pictorial Archive. Clay Fernald produced the cover layout with QuarkXPress 4.1 using Adobe’s ITC Garamond font.

Melanie Wang designed the interior layout, based on a series design by David Futato. This book was converted by Julie Hawks to FrameMaker 5.5.6 with a format conversion tool created by Erik Ray, Jason McIntosh, Neil Walls, and Mike Sierra that uses Perl and XML technologies. The text font is Linotype Birka; the heading font is Adobe Myriad Condensed; and the code font is LucasFont’s TheSans Mono Condensed. The illustrations that appear in the book were produced by Robert Romano and Jessamyn Read using Macromedia FreeHand MX and Adobe Photoshop CS. The tip and warning icons were drawn by Christopher Bing. This colophon was written by Sanders Kleinfeld.