Joe Casad

Fourth Edition

Sams Teach Yourself

TCP/IP

in 24 Hours

SAMS

Sams Teach Yourself TCP/IP in 24 Hours

Copyright © 2009 by Pearson Education, Inc.

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-0-672-32996-8
ISBN-10: 0-672-32996-4

Library of Congress Cataloging-in-Publication Data:

Casad, Joe, 1958-
Sams teach yourself TCP/IP in 24 hours / Joe Casad. — 4th ed.
p. cm.
Includes index.
ISBN-13: 978-0-672-32996-8 (pbk.)
ISBN-10: 0-672-32996-4 (pbk.)
1. TCP/IP (Computer network protocol) I. Title. II. Title: Teach yourself TCP/IP in 24 hours. III. Title: TCP/IP in 24 hours.
TK5105.585.C37 2009
005.7'1376—dc22
2008031826

Printed in the United States of America

First Printing September 2008

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Bulk Sales

Sams Publishing offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

U.S. Corporate and Government Sales
1-800-382-3419
corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact

International Sales
international@pearson.com

Editor-in-Chief
Mark Taub

Acquisitions Editor
Trina MacDonald

Development Editor
Michael Thurston

Managing Editor
Kristy Hart

Project Editor
Betsy Harris

Indexer
Lisa Stumpf

Proofreader
San Dee Phillips

Technical Editor
Ravi Prakash

Publishing Coordinator
Olivia Basegio

Book Designer
Gary Adair

Compositor
Nonie Ratcliff

Introduction

Welcome to *Sams Teach Yourself TCP/IP in 24 Hours, Fourth Edition*. This book provides a clear and concise introduction to TCP/IP for newcomers, and also for users who have worked with TCP/IP but would like a little more of the inside story. The fourth edition includes new material on recent developments in TCP/IP and offers a closer look at topics such as

- Firewalls
- Streaming
- Web services

You'll find new chapters on casting and streaming, web services, and the new Web, as well as several new sections throughout the book on recent developments in TCP/IP.

Does Each Chapter Take an Hour?

Each chapter is organized so that you can learn the concepts within one hour. The chapters are designed to be short enough to read all at once. In fact, you should be able to read a chapter in less than one hour and still have time to take notes and reread more complex sections in your one-hour study session.

How to Use This Book

The books in the *Sams Teach Yourself* series are designed to help you learn a topic in a few easy and accessible sessions. *Sams Teach Yourself TCP/IP in 24 Hours, Fourth Edition*, is divided into six parts. Each part brings you a step closer to mastering the goal of proficiency in TCP/IP.

- Part I, "TCP/IP Basics," introduces you to TCP/IP and the TCP/IP protocol stack.
- Part II, "The TCP/IP Protocol System," takes a close look at each of TCP/IP's protocol layers: the Network Access, Internet, Transport, and Application layers. You learn about IP addressing and subnetting, as well as physical networks and application services. You'll also learn about the protocols that operate at each of TCP/IP's layers.

- Part III, "Networking with TCP/IP," describes some of the devices, services, and utilities necessary for supporting TCP/IP networks. You learn about routing and network hardware, DHCP, DNS, and IPv6.
- Part IV, "TCP/IP Utilities," introduces some of the common utilities used to configure, manage, and troubleshoot TCP/IP networks. You learn about Ping, Netstat, FTP, Telnet, and other network utilities.
- Part V, "TCP/IP and the Internet," describes the world's largest TCP/IP network: the Internet. You learn about the structure of the Internet. You also learn about HTTP, HTML, XML, email, and Internet streaming.
- Part VI, "Advanced Topics," describes topics such as web services, messaging, the semantic web, and TCP/IP security. Part VI ends with a case study showing how the components of TCP/IP interact in a real working environment.

The concepts in this book, like TCP/IP itself, are independent of a system and descend from the standards defined in Internet Requests for Comment (RFCs).

How This Book Is Organized

Each hour in *Sams Teach Yourself TCP/IP in 24 Hours, Fourth Edition*, begins with a quick introduction and a list of goals for the hour. You can also find the following elements.

Main Section

Each hour contains a main section that provides a clear and accessible discussion of the hour's topic. You'll find figures and tables helping to explain the concepts described in the text. Interspersed with the text are special notes labeled By The Way? These notes come with definitions, descriptions, or warnings that help you build a better understanding of the material.

By the Way

> These boxes clarify a concept that is discussed in the text. A By The Way might add some additional information or provide an example, but they typically aren't essential for a basic understanding of the subject. If you're in a hurry, or if you want to know only the bare essentials, you can bypass these sidebars.

Q&A

Each hour ends with some questions designed to help you explore and test your understanding of the concepts described in the hour. Complete answers to the questions are also provided.

> **By the Way**
>
> Additionally, some hours include Workshops—exercises designed to help you through the details or give you practice with a particular task. You'll find them only in hours where a little real-world exploration will help build a better understanding of the material. Even if you don't have the necessary software and hardware to undertake some of the exercises in the Workshop, you might benefit from reading through the exercises to see how the tools work in a real network implementation.

Key Terms

Each hour includes a summary of important key terms that are introduced in the hour. The key terms are compiled into an alphabetized list at the end of each hour.

HOUR 20

Web Services

What you'll learn in this hour:

- **Web services**
- **XML**
- **SOAP**
- **WSDL**
- **Web transactions**

The technologies of the Web have led to a new revolution in software development. The web service architecture lets the programmer leverage the tools of the Web for complex tasks never envisioned by the creators of HTML. This hour examines the web services infrastructure. You'll also get a quick look at how e-commerce websites process web transactions.

At the completion of this hour, you will be able to

- Discuss the web service architecture
- Understand the role of XML, SOAP, and WSDL in the web service paradigm
- Describe how e-commerce websites process monetary transactions

Understanding Web Services

Now that almost every computer has a web browser, and web servers are widely understood, visionaries and software developers have been hard at work devising new ways to use the tools of the Web. In the old days, a programmer who wanted to write a network application had to create a custom server program, a custom client program, and a custom syntax or format for the two applications to exchange information. The effort of writing all this software was a huge expense of time and brain space, but with the rising importance of computer networking, the goals of data integration and centralized management was driving the demand client server applications. Network program interfaces existed of course—otherwise many of the classic applications described in this book would have never evolved—but network programming typically required some significant, high-priced coding at the network interface.

An easier solution that emerged over time is to use the existing tools, technologies, and protocols of the Web as a basis for creating custom network applications. This approach, which is supported by big companies such as IBM and Microsoft, as well as open source advocates and development tool vendors around the world, is known as the **web services architecture**.

The idea behind the web services architecture is that the web browser, web server, and TCP/IP protocol stack handle the details of networking so the programmer can concentrate of the details of the application. In recent years, this technology has outgrown the original vision of the Web as a manifestation of the global Internet. This web services architecture is regarded now as an approach to building any sort of network application, whether that application is actually connected to the Internet. Large and powerful vendors such as Sun, Microsoft, and IBM have invested enormous resources in building component infrastructures to support this web services vision.

The HTTP delivery system is only part of what we know as web services. Also significant is the arrival of component architectures that provide ready-made classes, functions, and programming interfaces for working within a web-based environment.

Web service applications are often used in situations that require a simple client connection to a server that maintains inventory or processes orders. For instance, a manufacturing company might use a web services program to place orders, track deliveries, and maintain up-to-date information on the contents of the warehouse.

Almost any big company has a need for software that tracks appointments, orders, and inventory. A web service framework is good for gluing together disparate services and transactions into a single, unified environment.

Figure 20.1 shows a complete web services scenario. On the front end (the left side of Figure 20.1), the programmer can take advantage of the preexisting web infrastructure, which handles data transmission and also provides a user interface through the web browser application on the client computer. On the back end, the programmer relies on the preexisting data storage system provided by an SQL database. The programmer is left to concentrate on the center section of Figure 20.1, where the ready-made components of the web services platform further simplify the task of programming.

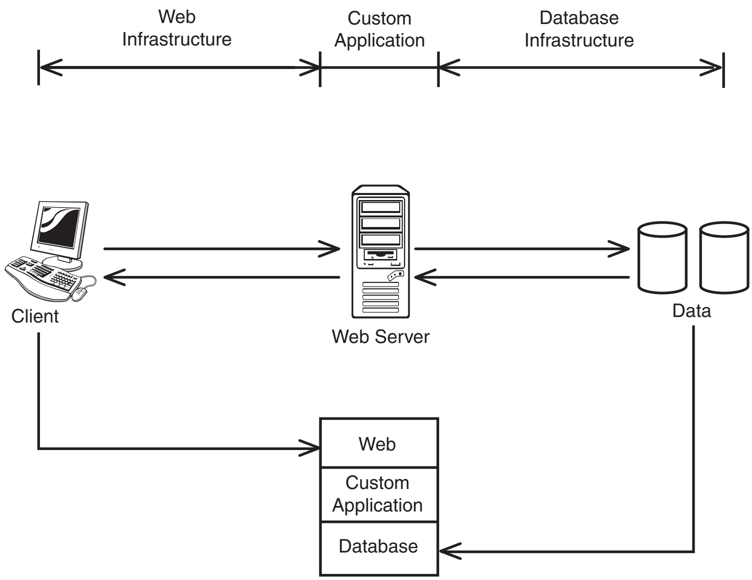

FIGURE 20.1
The web services programming model.

Data passes through the components of the web services system in XML format. XML is an efficient, universal means for assigning values to attributes. Experts quickly recognized that the system would work even better if they could use the XML format to actually invoke services or generate responses over the network. Simple Object Access Protocol (SOAP) offers a standard method for passing XML-based data between web service processes. SOAP also describes how to use the XML and HTTP to invoke remote procedures. As you learn later in this hour, SOAP messages pass to and from network services defined through the Web Services Description Language (WSDL).

XML

As soon as users, vendors, and web designers became accustomed to HTML, they started to ask for more. The growth of server-side and client-side programming techniques caused many experts to wonder if there might be a way to extend the rigid tag system of HTML. Their goal was to get beyond the conception of a markup language as a means for formatting text and graphics and to employ the language simply as a means for transmitting *data*. The result of this discussion was a new markup language called Extensible Markup Language, or XML.

As you learned earlier in this hour, the meaning and context for HTML data is limited to what you can express through a set of predefined HTML tags. If the data is enclosed in <H1> tags, it is interpreted as a heading. If the data is enclosed in <A> tags, it is interpreted as a link. XML, on the other hand, lets users define their own elements. The data can signify whatever you want it to signify, and you can invent the tag you will use to mark the data. For instance, if you follow horse racing, you could create an XML file with information on your favorite horses. That file might contain entries such as:

```
<horses>
     <horse_name="winky" breed="Thoroughbred">
         <sex="male" />
         <age="3" />
     </horse>
      <horse_name="Goddess" breed="Arabian">
         <sex="female" />
         <age="3" />
     </horse>
     <horse_name=""Gecko" breed="Uncertain">
         <sex="male" />
         <age="14" />
     </horse>
 </horses>
```

XML format looks a little like HTML, but it certainly isn't HTML. (Can you imagine how much your browser would choke if you tried to pass off <horse_name> as an HTML tag?) You can use whatever tag you want to use in XML, because you aren't preparing the data for some specific, rigidly predefined application like a web browser. The data is just data. The idea is that whoever creates the structure for the file will come along later to create an application or style sheet that will read the file and understand what the data means.

XML is an extremely powerful tool for passing data between applications. It is easy for a script or homegrown application to create XML as output or read XML as input. Even though a browser can't read XML directly, XML is still used extensively on the Web. In some cases, the XML data is generated on the server side and then converted to display-ready HTML before it is transmitted to the browser. Another technique is to provide an accompanying file called a Cascading Style Sheet (CSS) that tells how to interpret and display the XML data. However, XML is not limited to the web. Programmers now use XML for other contexts that require a simple, convenient format for assigning values to attributes.

XML now reaches far beyond the ordinary web as a format for storing and transmitting data. As long as the application that writes the XML data and the application that reads the data agree on the meaning of the elements, the data passes easily and economically between the applications through the miracle of XML.

By the Way

> XML is often described as a "markup language for creating markup languages."

SOAP

XML defines a universal format for exchanging application data. The universal XML specification alone, however, is not enough to provide developers with the infrastructure they need to create easy and elegant web services. Although XML provides an efficient format for reading and writing program data, XML alone does not provide a standard format for structuring and interpreting that data. The SOAP specification fills that role. SOAP is a standard protocol for exchanging XML-based messages that pass between the web-service client and server.

SOAP is designed to support communication between so-called SOAP nodes. (A SOAP node is basically a computer or application that supports SOAP.) The SOAP specification defines the structure of a message that passes from the SOAP sender to the SOAP receiver. Along the way, the message might pass through intermediate nodes that process the information in some way (see Figure 20.2). An intermediate node might provide logging, or it might modify the message somehow in transit to its final destination.

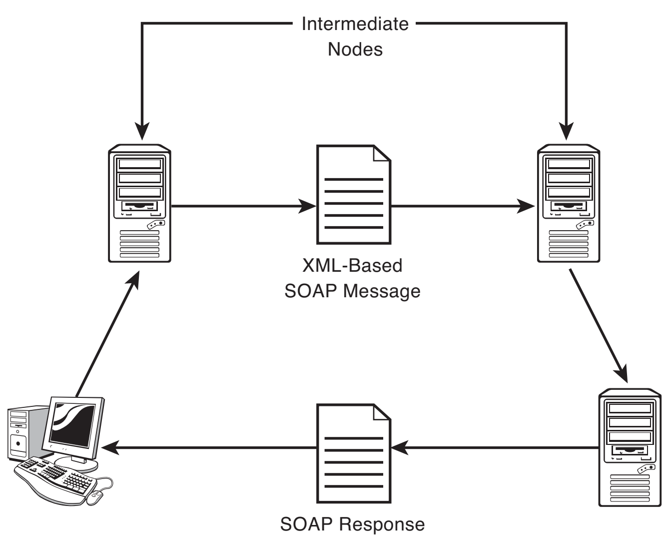

FIGURE 20.2
A SOAP message passes from the sender to the receiver and may pass through intermediate nodes.

At the conceptual level, a SOAP message from the client says "Here is some input. Process this and send me the output." The functionality of the application derives from a series of these XML-based SOAP messages in which the endpoints send information and receive responses. The formal structure of the SOAP message allows the software developer to easily create a SOAP-based client application that interacts with the server. For instance, a rental company that provides car rental reservations through a web-based server application could easily make the specifications available for a developer to write a custom client application that could connect to the server and reserve a car.

The structure of a SOAP message consists of an optional header and a message body. The header contains callouts, definitions, and meta-information that will be used by any node along the message path. The body includes data intended for the message recipient. For example, in the case of the car reservation service, the message body might contain data from the client describing the car the customer would like to rent and the date the vehicle must be available.

WSDL

The Web Services Description Language (WSDL) provides an XML format for describing the services associated with the web service application. According to the W3C's WSDL specification, "WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information." WSDL is a format for defining the services that exchange information through SOAP messages.

A WSDL document is primarily a set of definitions. The definitions within the document specify information on the data being transmitted and the operations associated with that data, as well as other data related to the service and the service location.

WSDL is not confined to SOAP but is also used with other web service communication protocols. In some cases, WSDL is used directly with HTTP to simplify the design and restrict the actions to more fundamental GET and POST-style operations at the heart of HTTP.

Web Service Stacks

Armed with XML, SOAP, WSDL, and the underlying components of TCP/IP and web service frameworks, a developer can easily create light and simple client and server applications that communicate through a web interface. Like TCP/IP itself, a web service environment consists of a stack of components. Major vendors have their own web service stacks that they provide to customers. The complete system forms a package of server software, developer tools, and even computer hardware that is provided to the client, along with consulting services and, sometimes, made-to-order custom applications.

Linux vendors and developers often talk about the LAMP stack, a collection of open source components that is easily tailored for web service environments. The memorable acronym LAMP spells out the principal components of the stack:

- **Linux**—An operating system that supports server applications running on the server system
- **Apache**—A web server that serves up XML-based SOAP messages
- **MySQL**—A database system that provides access to back-end data services
- **PHP (or Perl or Python)**—A web-ready programming language used to code the details of the custom web service application

Proprietary web service infrastructures provide similar features. The Java programming language is often used with web services—not just by Sun (the creators of Java), but also in IBM's WebSphere and other systems. Microsoft provides equivalents to Java through the tools of the .NET framework.

E-Commerce

An e-commerce site is not necessarily an implementation of the web service paradigm described earlier in this hour; however, it still might use some web-service techniques, especially on the back end. E-commerce is a high-profile example of the way applications and components can be combined together using the tools of the Web.

Vendors and advertisers began to notice early on that the Web is a great way to get people to buy things. It is no secret that many websites look like long, intricate advertisements. Despite the hype, which is enough to make anyone doubt the validity of the design, the fact is that the Web is a convenient and cost-effective way to shop. Rather than sending thousands of catalogs by direct mail, a vendor can simply post the catalog on the Web and let the customers find it through searches and links.

The business of buying over the Web did not really get started until vendors solved the security issues related to sending credit card information over the open Internet. In fact, Internet sales would not even be possible without the secure networking techniques. Most browsers are now capable of opening a secure communications channel with the server. This secure channel makes it impossible for a cyber thief to listen for passwords or credit card information.

A typical web transaction scenario is shown in Figure 20.3. The process is as follows:

1. A web server provides an online catalog accessible from the Web. A user browses through the product offerings from a remote location across the Internet.

2. The user decides to buy a product and clicks a Buy This Product link on the web page.

3. The server and browser establish a secure connection. (See Hour 23, "TCP/IP Security," for more on SSL and other secure communication techniques.) At this point, the browser sometimes displays a message that says something like "You are now entering a secure area…." Different browsers have different methods for indicating a secure connection.

4. After the connection is established, some form of authentication usually follows. On most transaction sites, the buyer establishes some form of user account with the vendor. This is partly for security reasons and partly for convenience (so the user can track the status of purchases). The user account information also lets the vendor track the behavior of the user and correlate the user's demographic information and purchase history. This logon step requires the web server to contact some form of back end database server—either to establish a new account or to check the credentials for logon to an existing account.

5. After the user is logged in, the server (or some application working on the server back end) must verify the credit card information and register the transaction with some credit card authority. Often this credit card authority is a commercial service affiliated with the credit card company.

6. If the transaction is approved, notice of the purchase and mailing information is transmitted to the vendor's fulfillment department, and the transaction application attends to the final details of confirming the purchase with the user and updating the user's account profile.

Operating system vendors such as Sun and Microsoft offer transaction server applications to assist with the important task of processing orders over the web. Because web transactions are highly specialized, and because they require an interface with existing applications on the vendor's network, application frameworks often provide special tools to assist with the task of constructing a transaction infrastructure.

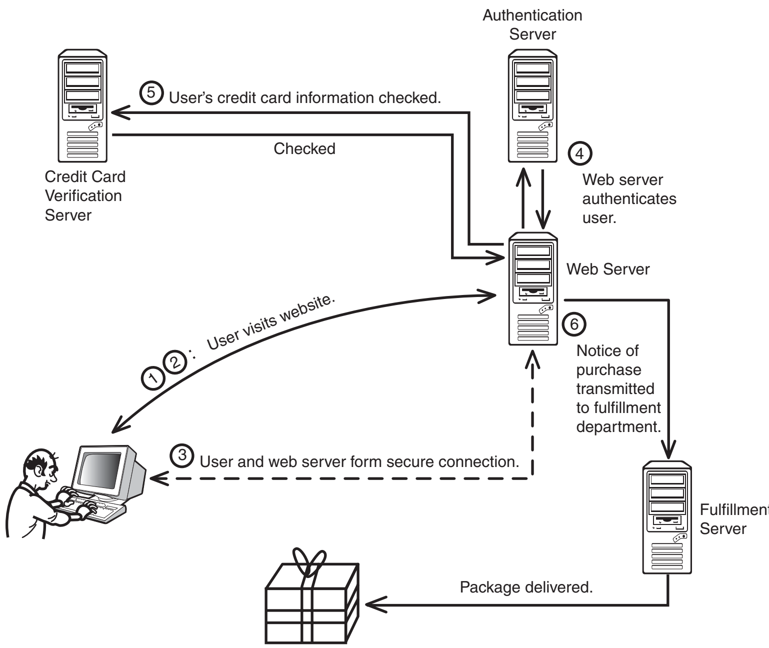

FIGURE 20.3
A typical web transaction scenario.

By the Way

Note that Figure 20.3 omits the role of the firewall within the transaction infrastructure. A large-scale commercial network might include a firewall behind the web server, protecting the network, and another firewall in front of the web server that blocks some traffic but leaves the server open to web requests. Also, on high-volume websites, you're more likely to find a collection of web servers sharing the load, rather than a single server.

Connections from the web server to the back-end servers could be across a protected internal network. Alternatively, the connection to the back end could be through a dedicated line that is separate from the main network. The credit card verification server is often an off-site service provided by a different company and accessed through a secure Internet connection.

Summary

The tools of the Web provide a backdrop for many kinds of application development. In addition to simple web pages and web forms, developers are putting together complex applications that place reservations, track inventory, and process purchase orders. This hour described some of the technologies at the heart of the web service paradigm. You learned about the web service infrastructure and why it is important. This hour also discussed three important web service components: XML, SOAP, and WSDL. Lastly, this hour took a look at the structure of web-based transactions.

Q&A

Q. ***What is the advantage of the web service model over conventional client-server programming?***

A. The web service model is design to integrate standard components that are already present on most networks, such as web server and web browser applications.

Q. ***Why is the web service model based on XML instead of HTML?***

A. HMTL is predefined collection of tags intended specifically as a markup language for web pages. XML has nearly unlimited capacity for defining new elements and assigning values to variables.

Q. ***Considering that countless vendors all have their own languages and components for supporting web services, what is the benefit of uniform standards like SOAP and WSDL?***

A. Standards like SOAP and WSDL provide a common format so that components written for different vendor environments can easily interact.

Key Terms

Review the following list of key terms:

- **LAMP**—An open source web service stack consisting of the Linux operating system, the Apache web server, the MySQL database system, and any of three programming languages that start with "P" (PHP, Perl, or Python).
- **SOAP**—A message exchange protocol for web applications.
- **Web service architecture**—A paradigm for building custom network applications around web components.
- **WSDL (Web Services Description Language)**—An XML-based format for describing network services.
- **XML (eXtensible Markup Language)**—A markup language used for defining and transmitting program data in a web service application.

Index

Symbols

A

E

F

J-K

L

Q-R

S