

Windows® Communication Foundation 4

John Sharp
content•master



online book + practice files

Step by Step



Microsoft®

Windows® Communication Foundation 4 Step by Step

John Sharp

Copyright © 2010 CM Group Ltd.

Complying with all applicable copyright laws is the responsibility of the user. All rights reserved. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without express written permission of the publisher.

Printed and bound in the United States of America.

ISBN: 978-0-735-64556-1

Third Printing: July 2014 with corrections November 2014

Microsoft, Microsoft Press, ActiveX, Excel, FrontPage, Internet Explorer, PowerPoint, SharePoint, Webdings, Windows, and Windows 7 are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other product and company names mentioned herein may be the trademarks of their respective owners.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

This book expresses the author's views and opinions. The information contained in this book is provided without any express, statutory, or implied warranties. Neither the author, Microsoft Corporation, nor their respective resellers or distributors, will be held liable for any damages caused or alleged to be caused either directly or indirectly by such information.

Acquisitions and Development Editor: Russell Jones

Production Editor: Kristen Borg

Production Services: Octal Publishing, Inc.

Technical Reviewer: Ashish Ghoda and Kenn Scribner

Indexing: Potomac Indexing, LLC

Cover: Karen Montgomery

Illustrator: Robert Romano

This product is printed digitally on demand.

Contents at a Glance

| | |
|--|-----|
| Chapter 1 | |
| Introducing Windows Communication Foundation | 1 |
| Chapter 2 | |
| Hosting a WCF Service | 45 |
| Chapter 3 | |
| Making Applications and Services Robust | 93 |
| Chapter 4 | |
| Protecting an Enterprise WCF Service | 121 |
| Chapter 5 | |
| Protecting a WCF Service over the Internet | 165 |
| Chapter 6 | |
| Maintaining Service Contracts and Data Contracts | 203 |
| Chapter 7 | |
| Maintaining State and Sequencing Operations | 243 |
| Chapter 8 | |
| Implementing Services by Using Workflows | 295 |
| Chapter 9 | |
| Supporting Transactions | 351 |
| Chapter 10 | |
| Implementing Reliable Sessions | 379 |
| Chapter 11 | |
| Programmatically Controlling the Configuration and Communications | 399 |
| Chapter 12 | |
| Implementing One-Way and Asynchronous Operations | 433 |

Chapter 13**Implementing a WCF Service for Good Performance 465****Chapter 14****Discovering Services and Routing Messages 491****Chapter 15****Building REST Services 547****Chapter 16****Using a Callback Contract to Publish and Subscribe to Events 599****Chapter 17****Managing Identity with Windows CardSpace 625****Chapter 18****Integrating with ASP.NET Clients and Enterprise Service
Components 647**

Table of Contents

| | |
|--|-----------|
| Acknowledgments | xi |
| Introduction | xiii |
| 1 Introducing Windows Communication Foundation | 1 |
| What Is Windows Communication Foundation? | 1 |
| The Early Days of Personal Computer Applications | 1 |
| Inter-Process Communications Technologies | 2 |
| The Web and Web Services | 3 |
| Using XML as a Common Data Format | 4 |
| Sending and Receiving Web Service Requests | 5 |
| JavaScript Object Notation and Rich Internet Applications | 5 |
| Handling Security and Privacy in a Global Environment | 6 |
| Service-Oriented Architectures and Windows Communication Foundation | 7 |
| Building a WCF Service | 9 |
| Defining the Contracts | 16 |
| Implementing the Service | 18 |
| Configuring and Testing the Service | 24 |
| Building a WCF Client Application | 30 |
| Deploying a WCF Service to Internet Information Services | 39 |
| WCF and the Principles of SOA | 42 |
| Summary | 43 |
| 2 Hosting a WCF Service | 45 |
| How Does a WCF Service Work? | 45 |
| Service Endpoints | 46 |
| Processing a Client Request | 47 |
| Hosting a WCF Service by Using Windows Process Activation Service | 49 |
| Hosting a Service in a User Application | 54 |
| Using the ServiceHost Class | 55 |

 **What do you think of this book? We want to hear from you!**

Microsoft is interested in hearing your feedback so we can continually improve our books and learning resources for you. To participate in a brief online survey, please visit:

www.microsoft.com/learning/booksurvey/

| | |
|---|------------|
| Building a Windows Presentation Foundation Application to Host a WCF Service | 58 |
| Reconfiguring the Service to Support Multiple Endpoints | 72 |
| Understanding Endpoints and Bindings | 76 |
| The WCF Predefined Bindings | 77 |
| Configuring Bindings | 81 |
| Default Endpoints | 82 |
| Hosting a WCF Service in a Windows Service | 84 |
| Summary | 91 |
| 3 Making Applications and Services Robust | 93 |
| CLR Exceptions and SOAP Faults | 94 |
| Throwing and Catching a SOAP Fault | 94 |
| Using Strongly Typed Faults | 99 |
| Reporting Unanticipated Exceptions | 110 |
| Managing Exceptions in Service Host Applications | 114 |
| ServiceHost States and Transitions | 114 |
| Handling Faults in a Host Application | 115 |
| Handling Unexpected Messages in a Host Application | 116 |
| Summary | 119 |
| 4 Protecting an Enterprise WCF Service | 121 |
| What Is Security? | 121 |
| Authentication and Authorization in a Windows Environment | 123 |
| Transport-Level and Message-Level Security | 124 |
| Implementing Security in a Windows Domain | 127 |
| Protecting a TCP Service at the Message Level | 127 |
| Protecting an HTTP Service at the Transport Level | 135 |
| Protecting an HTTP Service at the Message Level | 142 |
| Authenticating Windows Users | 146 |
| Authorizing Users | 152 |
| Using Impersonation to Access Resources | 160 |
| Summary | 163 |
| 5 Protecting a WCF Service over the Internet | 165 |
| Authenticating Users and Services in an Internet Environment | 166 |
| Authenticating and Authorizing Users by Using the ASP.NET Membership Provider and the ASP.NET Role Provider | 166 |

| | |
|--|------------|
| Authenticating and Authorizing Users by Using Certificates | 184 |
| Authenticating Service Messages by Using a Certificate | 195 |
| Summary | 202 |
| 6 Maintaining Service Contracts and Data Contracts | 203 |
| Modifying a Service Contract | 204 |
| Selectively Protecting Operations | 205 |
| Versioning a Service | 211 |
| Making Breaking and Nonbreaking Changes to a Service Contract | 222 |
| Modifying a Data Contract | 224 |
| Data Contract and Data Member Attributes | 224 |
| Data Contract Compatibility | 238 |
| Summary | 242 |
| 7 Maintaining State and Sequencing Operations | 243 |
| Managing State in a WCF Service | 244 |
| Service Instance Context Modes | 257 |
| Maintaining State with the PerCall Instance Context Mode | 262 |
| Selectively Controlling Service Instance Deactivation | 270 |
| Sequencing Operations in a WCF Service | 271 |
| Summary | 294 |
| 8 Implementing Services by Using Workflows | 295 |
| Building a Simple Workflow Service and Client Application | 296 |
| Implementing a Workflow Service | 296 |
| Implementing a Client Application for a Workflow Service | 313 |
| Handling Faults in a Workflow Service | 317 |
| Hosting a Workflow Service | 325 |
| Hosting a Workflow Service in IIS | 325 |
| Hosting a Workflow Service in a Custom Application | 328 |
| Implementing Common Messaging Patterns in a Workflow Service | 332 |
| Messaging Activities | 333 |
| Correlating Request and Reply Messages in a Workflow Service Instance | 335 |
| Using Messaging Activities to Implement Messaging Patterns | 336 |
| Managing Sessions and Maintaining State in a Workflow Service | 337 |
| Building Durable Workflow Services | 347 |
| Summary | 350 |

| | | |
|-----------|--|------------|
| 9 | Supporting Transactions | 351 |
| | Using Transactions in a WCF Service. | 352 |
| | Implementing OLE Transactions | 352 |
| | Implementing the WS-AtomicTransaction Protocol. | 369 |
| | Designing a WCF Service to Support Transactions. | 371 |
| | Transactions, Sessions, and Service Instance Context Modes | 371 |
| | Transactions and Messaging | 372 |
| | Transactions and Multi-Threading | 372 |
| | Implementing Transactions in a Workflow Service | 373 |
| | Long-Running Transactions. | 376 |
| | Summary | 377 |
| 10 | Implementing Reliable Sessions | 379 |
| | Using Reliable Messaging. | 380 |
| | Implementing Reliable Sessions with WCF | 381 |
| | Detecting and Handling Replay Attacks | 390 |
| | Configuring Replay Detection with WCF. | 391 |
| | Summary | 397 |
| 11 | Programmatically Controlling the Configuration and Communications | 399 |
| | The WCF Service Model | 399 |
| | Services and Channels. | 400 |
| | Behaviors. | 401 |
| | Composing Channels into Bindings | 403 |
| | Inspecting Messages. | 408 |
| | Controlling Client Communications | 419 |
| | Connecting to a Service Programmatically. | 419 |
| | Sending Messages Programmatically. | 427 |
| | Summary | 431 |
| 12 | Implementing One-Way and Asynchronous Operations | 433 |
| | Implementing One-Way Operations | 434 |
| | The Effects of a One-Way Operation | 434 |
| | One-Way Operations and Transactions. | 435 |
| | One-Way Operations and Timeouts. | 435 |
| | Implementing a One-Way Operation. | 436 |
| | Recommendations for Using One-Way Operations. | 445 |

| | |
|--|------------|
| Invoking and Implementing Operations Asynchronously | 446 |
| Invoking an Operation Asynchronously in a Client Application | 446 |
| Implementing an Operation Asynchronously in a WCF Service | 447 |
| Using Message Queues | 457 |
| Summary | 464 |
| 13 Implementing a WCF Service for Good Performance | 465 |
| Using Service Throttling to Control Resource Use | 466 |
| Configuring Service Throttling | 467 |
| Specifying Memory Requirements | 475 |
| Transmitting Data by Using MTOM | 476 |
| Sending Large Binary Data Objects to a Client Application | 478 |
| Controlling the Size of Messages | 484 |
| Streaming Data from a WCF Service | 487 |
| Enabling Streaming in a WCF Service and Client Application | 487 |
| Designing Operations to Support Streaming | 488 |
| Security Implications of Streaming | 490 |
| Summary | 490 |
| 14 Discovering Services and Routing Messages | 491 |
| Implementing Discovery | 491 |
| Configuring Ad Hoc Discovery | 492 |
| Handling Service Announcements | 499 |
| Using a Discovery Proxy | 507 |
| Implementing Routing | 523 |
| Routing Messages Manually | 524 |
| Using the RoutingService Class | 539 |
| Summary | 546 |
| 15 Building REST Services | 547 |
| Understanding the REST Model | 547 |
| Querying Data by Implementing a REST Web Service | 549 |
| Updating Data Through a REST Web Service | 570 |
| Using WCF Data Services | 579 |
| Consuming a WCF Data Service in a Client Application | 587 |
| Modifying Data by Using a WCF Data Service | 595 |
| Handling Exceptions in a Client Application | 597 |
| Summary | 598 |

| | |
|---|------------|
| 16 Using a Callback Contract to Publish and Subscribe to Events | 599 |
| Implementing and Invoking a Client Callback | 600 |
| Defining a Callback Contract | 600 |
| Implementing an Operation in a Callback Contract | 601 |
| Invoking an Operation in a Callback Contract | 604 |
| Reentrancy and Threading in a Callback Operation | 605 |
| Bindings and Duplex Channels | 606 |
| Using a Callback Contract to Notify a Client of the Outcome of a One-Way Operation | 606 |
| Using a Callback Contract to Implement an Eventing Mechanism | 614 |
| Delivery Models for Publishing and Subscribing | 620 |
| Summary | 623 |
| 17 Managing Identity with Windows CardSpace | 625 |
| Using Windows CardSpace to Access a WCF Service | 626 |
| Implementing Claims-Based Security | 626 |
| Using an Identity Provider | 641 |
| Claims-Based Authentication in a Federated Environment | 643 |
| Summary | 646 |
| 18 Integrating with ASP.NET Clients and Enterprise Services Components | 647 |
| Creating a WCF Service That Supports an ASP.NET Client | 647 |
| Exposing a COM+ Application as a WCF Service | 657 |
| Summary | 668 |
| Index | 669 |

 **What do you think of this book? We want to hear from you!**

Microsoft is interested in hearing your feedback so we can continually improve our books and learning resources for you. To participate in a brief online survey, please visit:

www.microsoft.com/learning/booksurvey/

Acknowledgments

On the back cover of his book, “Dirk Gently’s Holistic Detective Agency,” Douglas Adams depicts an invoice presented by Mr. Gently to his client for attempting to find her missing cat. It contains the following items:

| Item | Charge |
|---|-----------|
| Finding cat (deceased) | £50.00 |
| Detecting and triangulating vectors of interconnectedness of all things | £150.00 |
| Tracing same to beach on Bahamas, fare and accommodation | £1500.00 |
| Struggling on in face of draining skepticism from client, drinks | £327.00 |
| Saving the human race from total extinction | No charge |

Douglas Adams’s book was published in 1987, but 23 years later I find myself empathizing with Dirk Gently. Happily, my own beloved feline, Ginger, is still very much with us, but in common with many service-oriented developers these days, I spend more and more time searching for solutions that enable me to connect all things together. Clearly, my home office is not quite a beach on the Bahamas, but I do admit to enjoying a decent amount of time sunning myself in the stands at Edgbaston (the home of Warwickshire County Cricket Club) watching batsmen attempting to endanger workmen building the new pavilion with lofted drives over the boundary, while I contemplate how to configure pieces of software to get them to interoperate and communicate correctly. My wife is always a little skeptical of how our jaunts to see how Warwickshire fares against other county cricket teams amounts to “work,” but she enjoys the cricket as much as I do, so she does not complain.

In the world of connected solutions, Microsoft Windows Communication Foundation has proved an absolute boon, and although I am yet to be convinced that it has saved the human race from extinction, I have authored papers and even produced a video on how using WCF can help to save your organization (this may be hyperbole, but you know what we technophiles are like when we desperately want to convince management of the need to invest in new software and machinery!). To this end, I always count it an absolute privilege whenever I get the chance to write in depth about fun, new technology; as I mentioned in the previous edition of this book, I thank all at Content Master for allowing me to spend a significant amount of my time doing it.

It would also be very remiss of me not to thank Russell Jones at O’Reilly Media, who badgered me and patiently waited while I found the time to get started on this project as well as for all his support and help in editing and correcting my grammar during the initial drafts of each chapter, and to Bob Russell at Octal Publishing, who had the unrewarding job of having to wade through every chapter seeking out any remaining “British-isms.” Additionally,

sincere thanks are due to Ashish Ghoda and Kenn Scribner who took on the daunting task of checking the technical accuracy of my work and who provided valuable advice, guidance, and corrections whenever I was wrong (any remaining technical errors in the book are my own, of course). Also, thanks to Lin Joyner at Content Master, who expended a significant effort at great personal risk to her own sanity, for agreeing to test many of the exercises for me.

Finally, I must pay the greatest tribute to my long-suffering family: to Diana, my wife and fellow cricket-watcher, who is never short of advice when seeing a batsman struggle against a short-pitched delivery ("Just hit the ball!"); to James who grew up and left home to go to university while I was writing Chapter 17; and to Francesca who has learned from my wife what it takes to make a truly excellent cup of tea.

And to Ginger—please stop trying to walk across my keyboard as I type.

—John Sharp

Introduction

Microsoft Windows Communication Foundation (WCF), alongside Windows Workflow Foundation (WF) and Windows Presentation Foundation (WPF), has become part of the primary framework for building the next wave of business applications for the Microsoft Windows operating system. WCF provides the underpinning technology driving distributed solutions based on the Microsoft platform; with it, you can build powerful service-oriented systems designed to address connected services and applications. WCF is also an integral technology for building and accessing services running in the cloud under Windows Azure.

You can use WCF to create new services as well as augment and interoperate with many existing services created by using other technologies. When designing distributed applications in the past, you frequently had to choose a specific technology, such as Web services, COM+, Microsoft Message Queue, or .NET Framework Remoting. That choice often had a fundamental impact on the architecture of your solutions. In contrast, WCF provides a consistent model for implementing scalable and extensible systems that employ a variety of technologies with which you can design and architect your solutions without being restricted by a specific connectivity mechanism.

In short, if you are building professional, service-oriented solutions for Windows, you need to learn about WCF.

Who This Book Is For

This book will show you how to build connected applications and services using WCF. If you are involved in designing, building, or deploying applications for the Microsoft Windows operating system, sooner or later you are going to need to become familiar with WCF. This book will give you the initial boost you need to quickly learn many of the techniques required to create systems based on WCF. The book takes a pragmatic approach, covering the concepts and details necessary to enable you to build connected solutions.

Assumptions

To get the most from this book, you should meet the following profile:

- You should be an architect, designer, or developer who will be creating solutions for the Microsoft Windows family of operating systems.
- You should have experience developing applications using Visual Studio and C#.
- You should have a basic understanding of concepts such as transactions, Web services, security, and message queuing.

Finding Your Best Starting Point in This Book

This book is designed to help you build skills in a number of essential areas. It assumes that you are new to WCF and takes you step by step through the fundamental concepts of WCF, feature by feature. The techniques and ideas that you see in one chapter are extended by those in subsequent chapters; therefore, most readers should follow the chapters in sequence and perform each of the exercises. However, if you have specific requirements or are only interested in certain aspects of WCF, you can use the table below to find your best route through this book.

| If you are | Follow these steps |
|---|--|
| New to Web services and distributed applications and need to gain a basic understanding of WCF. | <ol style="list-style-type: none"> 1. Install the code samples as described in the “Code Samples” section of this Introduction. 2. Work through Chapters 1 to 5 sequentially and perform the exercises. 3. Complete Chapters 6 to 18 as your level of experience and interest dictates. |
| New to Web services and distributed applications and need to learn how to use WCF to implement solutions using common Web services features such as sessions, transactions, and reliable messaging. | <ol style="list-style-type: none"> 1. Install the code samples as described in the “Code Samples” section of this Introduction. 2. Work through Chapters 1 to 10 sequentially and perform the exercises. 3. Complete Chapters 11 to 18 as your level of experience and interest dictates. |
| Familiar with Web services and distributed applications, and need to learn about WCF quickly, including its advanced features. | <ol style="list-style-type: none"> 1. Install the code samples as described in the “Code Samples” section of this Introduction. 2. Skim the first chapter for an overview of WCF, but perform the exercises. 3. Read Chapter 2 and perform the exercises. 4. Skim Chapter 3. 5. Read Chapters 4 and 5 and complete the exercises. 6. Skim Chapters 6 to 10, performing the exercises that interest you. 7. Complete the remaining chapters and exercises. |
| Familiar with security concepts but need to understand how to use the security features that WCF provides. | <ol style="list-style-type: none"> 1. Install the code samples as described in the “Code Samples” section of this Introduction. 2. Skim the first three chapters. 3. Read Chapters 4 and 5 and perform the exercises. 4. Skim Chapters 6 to 15. 5. Read Chapter 17 and complete the exercises. 6. Skim Chapter 18. |

If you are

Referencing the book after working through the exercises.

Follow these steps

1. Use the index or the Table of Contents to find information about particular subjects.
2. Read the Summary sections at the end of each chapter to find a brief review of the concepts and techniques presented in the chapter.

Conventions and Features in This Book

This book presents information using conventions designed to make the information readable and easy to follow. Before you start, read the following list, which explains conventions you'll see throughout the book and points out helpful features that you might want to use:

- Each exercise is a series of tasks. Each task is presented as a series of numbered steps (1, 2, 3, and so on). A bullet (■) indicates an exercise that has only one step.
- Reader aids labeled "Note" and "Tip" provide additional information or alternative methods for completing a step successfully.
- Reader aids labeled "Important" alert you to information you need to check before continuing.
- Text that you type appears in **bold**.
- A plus sign (+) between two key names means that you must press those keys at the same time. For example, "Press Alt+Tab" means that you hold down the Alt key *while* you press the Tab key.

System Requirements

You'll need the following hardware and software to complete the practice exercises in this book:

- Microsoft Windows 7 Professional, Enterprise, or Ultimate editions.



Note Some of the exercises require you to create local users and security groups. This feature is not available with Windows 7 Home Basic or Home Premium editions.

- Microsoft Visual Studio 2010 Professional, Premium, Ultimate, or Test Professional editions, including SQL Server 2008 Express.



Note The exercises in this book are not intended to work with Visual Studio 2010 Express edition.

- 1.6 GHz or faster 32-bit (x86) or 64-bit (x64) processor.
- 1 GB RAM (32-bit) or 2 GB RAM (64-bit).
- 20 GB available hard disk space (32-bit) or 25 GB (64-bit).
- DirectX 9 graphics device with WDDM 1.0 or higher driver.
- Microsoft mouse (or compatible) pointing device.

Some of the exercises require that you have installed Internet Information Services (IIS) and Message Queuing (MSMQ). You will also need the *AdventureWorks* database provided with the code samples for this book. Download and installation instructions are provided later in this introduction.



Important If you have other tools or services that establish network connections, you may need to temporarily halt them if they use the same ports required by the exercises in this book (alternatively, you can replace the port numbers referenced by the exercises with others of your own choice). For example, some of the exercises reference port 8080. If you have the Apache Web server running on your development computer, it defaults to using port 8080, so you may need to halt or reconfigure this service.

Code Samples

Follow these steps to download and install the code samples and other companion content on your computer so that you can use them with the exercises:

1. Navigate to <http://aka.ms/645561/files>.
2. Click the Downloads tab.

You'll see instructions for downloading the .zip archive containing the companion content files.

3. Unpack the .zip archive into your Documents folder. This action creates the following folder containing the exercise and solution files for each chapter:

Microsoft Press\WCF Step By Step

Installing and Configuring Internet Information Services and Microsoft Message Queue

Many of the exercises in this book require you to build WCF services hosted by using Internet Information Services (IIS). You must make sure that you have installed and configured IIS on your computer, and you must have installed ASP.NET version 4.0 with IIS. Additionally, some exercises use Microsoft Message Queue (MSMQ) as the transport for connecting client applications to services, so you must also install the MSMQ Server Core. The following instructions describe how to do this. Note that you require administrative access to your computer to install and configure IIS and MSMQ.

1. Log on to Windows as an account that has Administrator access.
2. On the Windows Start menu, click Control Panel, and then click Programs. In the Programs pane, under Programs And Features, click Turn Windows Features On Or Off.
3. In the Windows Features dialog box, expand Internet Information Services, and then select the following features:
 - ❑ Web Management Tools | IIS Management Console
 - ❑ Web Management Tools | IIS 6 Management Compatibility | IIS 6 Metabase and IIS 6 Configuration Compatibility
 - ❑ World Wide Web Services | Application Development Features | ASP.NET (this will also select .NET Extensibility, ISAPI Extensions, and ISAPI Filters)
 - ❑ World Wide Web Services | Common Http Features | Directory Browsing (Default Document should already be selected)
 - ❑ World Wide Web Services | Security | Basic Authentication and World Wide Web Services | Security | Windows Authentication (Request Filtering should already be selected)
4. Expand Microsoft Message Queue (MSMQ) Server, and then select Microsoft Message Queue (MSMQ) Server Core (do not select the individual items in the Microsoft Message Queue (MSMQ) Server Core folder).
5. Click OK, and then wait for the features to be installed and configured.

Installing ASP.NET Version 4.0

The exercises in this book rely on ASP.NET Version 4.0 being installed and configured with IIS. To do this, perform the following tasks:

1. On the Windows Start menu, click All Programs, click Microsoft Visual Studio 2010, click Visual Studio Tools, right-click Visual Studio Command Prompt (2010), and then click Run As Administrator. In the User Account Control dialog box, click Yes.
2. In the Visual Studio Command Prompt window, type the following command:

```
aspnet_regiis -iru
```
3. When the command has completed, leave the Visual Studio Command Prompt window open; you will use it again after installing the *AdventureWorks* database.

Installing and Configuring the AdventureWorks Database

The exercises and examples in this book make use of the *AdventureWorks* sample database. If you don't already have this database installed on your computer, a copy of the database installation program is supplied with the companion content for this book. Follow these steps to install and configure the database:

1. Log on to Windows as an account that has administrator access if you have not already done so.
2. Verify that the SQL Server (SQLEXPRESS) service is running.



Tip Start the SQL Configuration Manager utility in the Configuration Tools folder, located in the Microsoft SQL Server 2008 program group. In the left pane, click SQL Server Services. In the right pane, examine the status of the SQL Server (SQLEXPRESS) service. If the status is Stopped, right-click the service, and then click Start. Wait for the status to change to Running, and then close SQL Configuration Manager.

3. Using Windows Explorer, move to the Microsoft Press\WCF Step By Step\Setup folder located within your Documents folder.
4. Double-click the file *AdventureWorks2008_SR4.exe*. If the User Account Control dialog box appears, click Yes.
5. Wait while the WinZip Self-Extractor tool unzips the installation program.
6. When the SQL Server 2008R2 Database Installer dialog box appears, read the license agreement. If you agree with the license terms, select the I Accept The License Terms check box, and then click Next.

7. On the AdventureWorks 2008 Community Sample Database SR4 page, set the Installation Instance to SQLEXPRESS, select the *AdventureWorks OLTP* database, deselect all other databases, and then click Install.



Note Make sure that you select the AdventureWorks OLTP database and *not* AdventureWorks OLTP 2008. Depending on how you have configured SQL Server, not all databases will be available anyway, and you may see a warning icon against some of these databases. You can ignore these warnings because these databases are not required.

8. On the Installation Execution page, wait while the database is installed and configured, and then click Finish.
8. Return to the Visual Studio Command Prompt window running as Administrator in the Microsoft Press\WCF Step By Step\Setup folder.
9. Type the following command:

```
osql -E -S .\SQLEXPRESS -i aspnet.sql
```

This command should complete without any errors (it will display a series of prompts, "1> 2> 1> 2> 1> 2> 1> 1> 2> 1> 2> 1>").



Note The script *aspnet.sql* creates user accounts for the DefaultAppPool and ASP.NET v4.0 applications pools used by IIS and grants these accounts access to the *AdventureWorks* database.

10. Close the Visual Studio Command Prompt window.

Using the Code Samples

Each chapter in this book explains when and how to use any code samples for that chapter. When it's time to use a code sample, the book will list the instructions for how to open the files. The chapters are built around scenarios that simulate real programming projects, so you can easily apply the skills you learn to your own work.

For those of you who like to know all the details, following is a list of the code sample, Visual Studio projects, and solutions, grouped by the folders where you can find them.



Important Many of the exercises require administrative access to your computer. Make sure you perform the exercises using an account that has this level of access.

| Solution Folder | Description |
|---|---|
| Chapter 1 | |
| Completed\ProductsService | <p>This solution gets you started. Creating the ProductsService project leads you through the process of building a simple WCF service hosted by IIS. You can use the service to query and update product information in the <i>AdventureWorks</i> database.</p> <p>The ProductsClient project is a console-based WCF client application that connects to the <i>ProductsService</i> service. You use this project for testing the WCF service.</p> |
| Chapter 2 | |
| ProductsClient | <p>This solution is the starting point for the exercises in this chapter. It contains a copy of the completed client application from Chapter 1.</p> |
| Completed\ProductsClient | <p>This solution contains a version of the client application that connects to the <i>ProductsService</i> service by using a TCP connection.</p> |
| Completed\HostedProductsServiceHost | <p>This solution contains Windows Presentation Foundation application that provides a host environment for the <i>ProductsService</i> service. You use this application to manually start and stop the service. You configure the ProductsClient application to connect to the service hosted by this application by using an HTTP endpoint.</p> |
| Completed\WindowsProductService | <p>This solution contains a Windows Service that hosts the <i>ProductsService</i> service. You can start and stop the service from the Services applet in the Windows Control Panel. You reconfigure the ProductsClient application to connect to this service by using an endpoint based on the Named Pipe transport.</p> |
| Chapter 3 | |
| ProductsServiceFault | <p>This solution contains a copy of the ProductsServiceLibrary, ProductsServiceHost, and ProductsClient applications from Chapter 2. It is used as a starting point for the exercises in this chapter.</p> |
| Completed\UntypedProductsServiceFault | <p>The <i>ProductsService</i> service in this solution traps exceptions and reports them back to the client application as untyped SOAP faults, which are caught and handled by the ProductsClient application.</p> |
| Completed\StronglyTypedProductsServiceFault | <p>The <i>ProductsService</i> service in this solution reports exceptions as typed SOAP faults, defined by using fault contracts. The ProductsClient application catches these strongly typed SOAP faults as exceptions.</p> |
| Chapter 4 | |
| ProductsService | <p>This solution contains three projects: the <i>ProductsService</i> service, the ProductsServiceHost application, and the ProductsClient. These projects are configured to catch and handle SOAP faults, as described in Chapter 3. This solution forms the starting point for the exercises in this chapter.</p> |

| Solution Folder | Description |
|--|--|
| Chapter 4 (continued) | |
| Completed\NetTcpProductsServiceWithMessageLevelSecurity | This solution contains an implementation of the <i>ProductsService</i> service, the <i>ProductsServiceHost</i> application, and the <i>ProductsClient</i> applications that applies message-level security over a TCP binding. |
| Completed\BasicHttpProductsServiceWithTransportLevelSecurity | This solution shows how to implement transport-level security over an HTTP binding. |
| Completed\WS2007HttpProductsServiceWithMessageLevelSecurity | This version of the solution contains a host that implements message-level security over an HTTP binding. |
| Completed\ProductsServiceWithBasicAuthentication | This solution contains a version of the <i>ProductsService</i> service that implements basic authentication and displays the name of the user calling the <i>ListProducts</i> operations. The client application explicitly provides the name and password of the user connecting to the service. |
| Completed\ProductsServiceWithWindowsAuthentication | This solution is similar to the previous one, except that the <i>ProductsService</i> service implements Windows authentication. The credentials for the client application are picked up from the user's login session. |
| Completed\ProductsServiceWithAuthorization | The <i>ProductsService</i> service in this solution authorizes users according to the Windows security group to which they belong. Users that do not belong to a specified security group are denied access when they attempt to invoke operations. |
| Chapter 5 | |
| ProductsClient | This folder contains a copy of the client application that is used for testing the various configurations of the <i>InternetProductsService</i> service in this chapter. |
| Completed\ASPNETMembership | This solution contains the <i>InternetProductsService</i> service that is deployed to IIS and authenticates users by using the ASP.NET Role Provider rather than Windows security groups. |
| Completed\ASPNETMemberShipUsingCertificates | The <i>InternetProductsService</i> service in this solution uses the ASP.NET Role Provider in conjunction with certificates to authenticate users. |
| Completed\MutualAuthenticationUsingCertificates | The <i>InternetProductsService</i> service in this solution uses a certificate to authenticate itself to the client application. |
| Chapter 6 | |
| ProductsService | This solution contains an amended copy of the <i>ProductsClient</i> , <i>ProductsServiceLibray</i> , and <i>ProductsServiceHost</i> projects from Chapter 4. The service implements message-level security and authenticates users by using Windows tokens. This solution is used as the starting point for the exercises in this chapter. |

| Solution Folder | Description |
|---|---|
| Chapter 6 (continued) | |
| ProductsServiceWithVersionedServiceContract | This solution contains an implementation of the <i>ProductsService</i> service and a client application that provides these two versions of the service contract. It is used by some of the exercises in the second part of the chapter. |
| Completed\ProductsServiceWithProtectedOperations | This solution contains a version of the <i>ProductsService</i> service in which client applications are required to encrypt and sign request messages for some operations, but only sign requests for others. The proxy class in the <i>ProductsClient</i> application has been updated to encrypt and sign, or just sign messages, as appropriate. The purpose of this solution is to show how changing security requirements for operations can break a service contract. |
| Completed\ProductsServiceWithAdditionalBusinessLogic | The <i>ProductsService</i> service in this solution contains additional methods. However, because these methods implement internal logic for the service and are not exposed as part of the service contract, they do not require that existing client applications are updated. |
| Completed\ProductsServiceWithModifiedServiceContract | This solution contains a version of the <i>ProductsService</i> service with an additional operation and a modified service contract. The client application has not been updated, but it still works although it cannot invoke the new operation. |
| Completed\ProductsServiceWithVersionedServiceContract | The <i>ProductsService</i> service in this solution exposes two versions of the service contract, enabling existing client applications to use the old contract while exposing the additional operation to new client applications. |
| Completed\ProductsServiceWithAdditionalFieldsInDataContract | This solution shows the effects that modifying a data contract can have on client applications and how you can implement a data contract that supports clients expecting different versions of a data contract. |
| Chapter 7 | |
| Completed\ShoppingCart | This solution contains a completed version of the initial <i>ShoppingCartService</i> service that implements shopping cart functionality and a client application that exercises this functionality. This solution is used as the basis for subsequent exercises in this chapter. |
| Completed\ShoppingCartContextModes | The <i>ShoppingCartService</i> service in this solution demonstrates the use of the Single instance context mode. |
| Completed\ShoppingCartWithState | The <i>ShoppingCartService</i> service in this solution uses the <i>PerCall</i> instance context mode and contains code that saves the instance state to XML files. |
| Completed\ShoppingCartWithSequencedOperations | This solution shows how to control the sequence in which a client application can invoke operations and control the lifetime of a session. |

| Solution Folder | Description |
|---|---|
| Chapter 7 (continued) | |
| DurableShoppingCart | This solution contains a version of the <i>ShoppingCartService</i> service that implements the <i>PerSession</i> instance context mode. The solution also contains a GUI client application called Shopping CartGUIClient. This solution is used by exercises that convert the <i>ShoppingCartService</i> service into a durable service. |
| Completed\DurableShoppingCart | This solution contains a completed implementation of the durable version of <i>ShoppingCartService</i> service. |
| Chapter 8 | |
| Completed\ProductsWorkflow | This solution contains a workflow service called <i>ProductsWorkflowService</i> that retrieves the details of a specified product. The solution also includes a basic console client application to test the service. |
| Completed\ProductsWorkflow WithFaultHandling | The <i>ProductsWorkflowService</i> service in this solution shows how to catch exceptions in a service and send SOAP faults to a client application. |
| ProductsClient | This version of the client application for the <i>ProductsWorkflowService</i> service that generates SOAP faults. |
| Completed\ProductsWorkflow WithIISDeployment | This solution shows how to deploy the <i>ProductsWorkflowService</i> service to IIS. |
| Completed\ProductsWorkflow WithCustomHost | This solution demonstrates how to create a custom host application for a workflow service. |
| Completed\ShoppingCartService | This solution contains a completed version of the <i>ShoppingCartService</i> service implemented as a workflow service. |
| ShoppingCartGUIClient | This is a copy of the Shopping CartGUIClient developed in Chapter 7. It is used to test the workflow version of the <i>ShoppingCartService</i> service. |
| Completed\ShoppingCartWith HostAndClient | This solution contains a complete version of the workflow version of the <i>ShoppingCartService</i> service, hosted in a custom host application and accessed from the Shopping CartGUIClient application. |
| Completed\DurableShopping CartWithHostAndClient | This solution demonstrates how to implement a workflow service as a durable service. |
| Chapter 9 | |
| ShoppingCartService | This solution contains a copy of the non-durable Shopping Cart Service, Shopping CartServiceHost, and Shopping CartClient projects from Chapter 7. It is used as the starting point for the exercises in this chapter. |

| Solution Folder | Description |
|---|--|
| Chapter 9 (continued) | |
| Completed\ShoppingCartService | This solution contains a version of the <i>ShoppingCartService</i> service that uses transactions to maintain database integrity. The client application initiates the transactions. |
| ProductsWorkflow | This solution shows how to implement transactions in a workflow service. It is based on the <i>ProductsWorkflowService</i> from Chapter 8. The client application is also based on a workflow. |
| Chapter 10 | |
| ShoppingCartService | This solution contains a completed version of the Shopping CartService, ShoppingCartHost, and ShoppingCartClient applications from Chapter 9. It is used as the starting point for the exercises in this chapter. |
| Completed\ShoppingCartService | This solution shows how to configure the <i>ShoppingCartService</i> service and ShoppingCartClient application to implement reliable sessions. You run the client application and use the WCF Service Trace Viewer utility to examine the messages passing between the client application and service. |
| Completed\ShoppingCartServiceWithReplayDetection | This solution implements a custom binding for the <i>ShoppingCartService</i> service and ShoppingCartClient applications to support the secure conversation protocol and provide automatic message replay detection. |
| Chapter 11 | |
| ShoppingCartService | This solution contains a copy of the completed ShoppingCart Service and ShoppingCartClient projects from Chapter 10. The binding and endpoint configuration has been removed from the ShoppingCartHost project. In the exercises in this chapter, you implement these items in code rather than by providing them in a configuration file. |
| Completed\ShoppingCartService | This solution contains an implementation of the ShoppingCartHost application that programmatically creates a custom binding rather than using one of the WCF predefined bindings. |
| Completed\ShoppingCartServiceWithMessageInspector | This solution shows how to create a custom service behavior with which you can inspect request messages sent to the service and response messages that it sends back to client applications. |
| ProductsService | This solution contains a copy of the <i>ProductsService</i> service from Chapter 6. The code and configuration information in the client that connects to the service and sends request messages has been removed. You add code that performs these tasks programmatically in the exercises in this chapter. |
| Completed\ProductsService | This solution contains a completed version of the ProductsClient application. The client application connects to the service by creating a binding and a channel programmatically rather than using a generated proxy class. |

| Solution Folder | Description |
|----------------------------------|--|
| Chapter 11 (continued) | |
| ProductsServiceWithManualProxy | This solution shows how to inherit from the <i>ClientBase</i> generic abstract class to implement a proxy class that enables a client application to authenticate itself to the service. |
| SimpleProductsService | This solution contains a stripped down version of the <i>Products Service</i> service and client application. You add code to the client application to connect to the service by creating a binding and a channel and then manually create and send a SOAP message to the service. |
| Completed\SimpleProducts Service | This solution contains a completed version of the client application that manually creates and sends a SOAP message to the service. It receives the response also as a SOAP message. |
| Chapter 12 | |
| Completed\OneWay | This solution contains a new service called <i>AdventureWorksAdmin</i> . The <i>AdventureWorksAdmin</i> service exposes an operation that can take significant time to run. It demonstrates how to implement this operation as a one-way operation. You also use this solution to understand the circumstances under which a one-way operation call can block a client application and how to resolve this blocking. |
| Completed\Async | This solution contains a version of the <i>AdventureWorksAdmin</i> service that implements an operation that can execute asynchronously. |
| MSMQ | This solution contains a copy of the <i>AdventureWorksAdmin</i> service that acts as the starting point for the exercises that demonstrate how to use MSMQ as the transport for a WCF service. |
| Completed\MSMQ | This version of the solution contains a completed implementation of the <i>AdventureWorksAdmin</i> service that uses a message queue to receive messages from client applications. You run the client application and service at different times and verify that messages sent by the client application are queued and received when the service runs. |
| Chapter 13 | |
| Throttling | This solution contains a simplified, non-transactional version of the <i>ShoppingCartService</i> service and an extended version of the client application that simulates multiple users connecting to the service. This solution provides the starting point for the exercises showing how to implement throttling. |
| Completed\Throttling | This solution contains the completed version of the <i>Shopping CartService</i> service. You use this service to test the way in which you can configure WCF to conserve resources during periods of heavy load. |

| Solution Folder | Description |
|---|--|
| Chapter 13 (continued) | |
| MTOM | This solution contains a service called <i>ShoppingCartPhotoService</i> that retrieves photographic images of products from the <i>Adventure Works</i> database. The solution also contains a basic WPF client application that displays images sent by the server. You use this solution to examine how a WCF service transmits messages containing large amounts of binary data. |
| Completed\MTOM | This version of the service encodes the binary data constituting the image by using the Message Transmission Optimization Mechanism (MTOM). You use this solution to generate message traces that you examine so you can see how the messages are encoded. |
| Streaming | This solution contains a version of the <i>ShoppingCartPhotoService</i> that uses streaming to send the image data to the client application rather than MTOM. |
| Chapter 14 | |
| ProductsService | This solution contains a copy of the <i>ProductsService</i> service hosted by the ASP.NET Development Web Server, and client application that connects to this service. This solution is used as the starting point for the exercises that show how to implement service discovery. |
| Completed\ProductsServiceWithAdHocDiscovery | The <i>ProductsService</i> service in this solution implements ad hoc discovery. It is deployed to IIS. The client application is modified to broadcast a discovery request and retrieve the address of the <i>ProductsService</i> service. |
| Completed\ProductsServiceWithAnnouncements | In this version of the solution, the <i>ProductsService</i> service sends announcement messages when it starts up and shuts down. The client application listens for service announcements and caches the URLs of services as they come on-line. When the client application sends a request, it looks up the URL of the service in this cache. |
| Completed\ProductsServiceWithManagedDiscovery | This solution shows how to implement a discovery proxy. The <i>ProductsService</i> service sends announcement messages, and the discovery proxy listens for these messages and caches the URLs of services as they come on-line. The client application is modified to retrieve the address of the <i>ProductsService</i> from the discovery proxy. |
| LoadBalancingRouter | This solution contains an amended copy of the durable <i>ShoppingCartService</i> , <i>ShoppingCartServiceHost</i> , and <i>ShoppingCartGUIClient</i> from Chapter 7. It is used as the basis for the exercises that show how to implement routing inside a WCF service. |
| Completed\LoadBalancingRouter | This solution contains a WCF service called <i>ShoppingCartServiceRouter</i> that acts as a load-balancing router for two instances of the <i>ShoppingCartService</i> service. The client application connects to the router, which transparently redirects requests to one instance or the other of the <i>ShoppingCartService</i> service. |

| Solution Folder | Description |
|---|---|
| Chapter 14 (continued) | |
| ShoppingCartServiceWithRouter | This solution contains another copy of the durable Shopping CartService, ShoppingCartServiceHost, and ShoppingCartGUI Client from Chapter 7, except that the host application is preconfigured with two HTTP endpoints. This solution provides the starting point for the exercises that show how to implement a WCF routing service. |
| Completed\ShoppingCartServiceWithRouter | This solution contains a completed implementation of the WCF routing service |
| MessageInspector | This solution contains a version of the <i>MessageInspector</i> behavior created in Chapter 11. It is used to test the routing service in the ShoppingCartServiceWithRouter project by displaying the details of messages as they are received by the ShoppingCartHost project. |
| Chapter 15 | |
| Completed\ProductsSales | This solution contains a REST Web service called <i>ProductsSales Service</i> , host, and client application that provides access to sales information. The client application tests the <i>ProductsSalesService</i> service by sending requests that query the details of orders and customers. |
| Completed\ProductsSalesWithUpdates | This solution contains an updated version of the <i>ProductsSales Service</i> service that supports insert, update, and delete operations. The client application is extended to test this functionality. |
| Completed\SalesData | This solution contains a REST Web service called <i>SalesData</i> that also provides access to customer and order information. This service is implemented by using the WCF Data Services template. The SalesDataClient application in this solution uses the client library for the service to connect and send requests to the service. |
| Chapter 16 | |
| ProductsServiceV3 | This solution contains another version of the <i>ProductsService</i> service that provides an additional operation that updates the price of a product. The solution also contains a host application, and a client application for testing the service. |
| Completed\ProductsServiceV3 | In this solution, the <i>ProductsService</i> service implements a callback contract. The operation that changes the price of a product is reconfigured as a one way operation, and the callback contract enables the service to asynchronously notify the client application of the result of the operation when it has completed. |
| Completed\ProductsServiceV3WithEvents | This version of the <i>ProductsService</i> service implements an eventing mechanism. Instances of the client applications subscribe to an event, and the service uses a callback contract to notify each subscribing client when the event occurs. |

| Solution Folder | Description |
|-------------------------------|---|
| Chapter 17 | |
| ShoppingCartService | This solution contains a completed version of the <i>ShoppingCart Service</i> service, host, and client applications from Chapter 10. It is used as the starting point for the exercises in this chapter. |
| Completed\ShoppingCartService | The <i>ShoppingCartService</i> service in this solution implements claims-based security. The client application uses Windows CardSpace to manage user credentials and send claims information to the service. The service uses verified claims to authorize access to users. |
| Chapter 18 | |
| ASPNETService | This solution contains a legacy ASP.NET Web site called ASPNET ProductsService. This Web site provides an ASP.NET Web service. The solution also contains a client application that connects to this Web service. Both applications were developed by using the .NET Framework 2.0. The service is used as the basis for exercises that show how to migrate an ASP.NET Web service to WCF and the .NET Framework 4.0. |
| ProductsServiceHost | This project contains the host application for the WCF service that implements the functionality migrated from the <i>ASPNETService</i> Web service. |
| Completed\ASPNETService | This solution is a version of the <i>ASPNETProductsService</i> service that has been migrated to WCF, together with the host and client applications. The code in the client application has not changed, and connects to WCF service in exactly the same way as it did to the original ASP.NET Web service. |
| Products | This solution contains a legacy COM+ application that you configure to appear to client applications as a WCF service. |
| ProductsClient | This solution contains an incomplete copy of the ProductsClient application for testing the Products COM+ application by connecting to it as though it was a WCF service. You finish this application during the exercises in this chapter. |
| Completed\ProductsClient | This solution contains the completed version of the ProductsClient application. |

Uninstalling the Code Samples

To remove the code samples from your computer, delete the folder Microsoft Press\WCF Step By Step from your Documents folder by using Windows Explorer.

Your Companion eBook

The eBook edition of this book allows you to:

- Search the full text
- Print
- Copy and paste

To download your eBook, please see the instruction page at the back of this book.

Errata and Book Support

We've made every effort to ensure the accuracy of this book and its companion content. If you do find an error, please report it on our Microsoft Press site:

1. Go to <http://microsoftpressstore.com>.
2. In the Search box, enter the book's ISBN or title.
3. Select your book from the search results.
4. On your book's catalog page, find the Errata and Updates tab.

You'll find additional information and services for your book on its catalog page. If you need additional support, please e-mail Microsoft Press Book Support at mspinput@microsoft.com.

Please note that product support for Microsoft software is not offered through the addresses above.

We Want to Hear from You

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at:

<http://www.microsoft.com/learning/booksurvey>

The survey is short, and we read *every one* of your comments and ideas. Thanks in advance for your input!

Stay in Touch

Let's keep the conversation going! We're on Twitter: <http://twitter.com/MicrosoftPress>.

This page intentionally left blank

This page intentionally left blank

Chapter 3

Making Applications and Services Robust

After completing this chapter, you will be able to:

- Explain how the WCF runtime can convert common language runtime exceptions into SOAP fault messages to transmit exception information from a WCF service.
- Use the *FaultContract* attribute in a service to define strongly typed exceptions as SOAP faults.
- Catch and handle SOAP faults in a client application.
- Describe how to configure a WCF service to propagate information about unanticipated exceptions to client applications for debugging purposes.
- Describe how to detect the *Faulted* state in a WCF service host application and how to recover from this state.
- Explain how to detect and log unrecognized messages sent to a service.

Detecting and handling exceptions is an important part of any professional application. In a complex desktop application, many different situations can raise an exception, ranging from programming errors or events such as unexpected or malformed user input, to failure of one or more hardware components in the computer running the application. In a distributed environment, the scope for exceptions is far greater. This is due to the nature of networks and the fact that, in some cases, neither the application nor the development or administrative staff has control over how the network functions or its maintenance (who is responsible for making sure that the Internet works?). If you factor in the possibility that your application might also access services written by some third party, who may modify or replace the service with a newer version (possibly untested!), or remove the service altogether, then you might begin to wonder whether your distributed applications will ever be able to work reliably.

This chapter shows you how to handle exceptions in client applications and services developed using WCF. You will see how to specify the exceptions that a WCF service can raise and how to propagate information about exceptions from a WCF service to a WCF client. You will also explore the states that a service can be in, how to determine when a host application switches from one state to another, and how to recover a service that has failed. Finally, you will see how to detect unrecognized messages sent to a service by client applications.

CLR Exceptions and SOAP Faults

A WCF service is a managed application that runs by using the .NET Framework common language runtime, or CLR. One important feature of the CLR is the protection that it provides when an error occurs; the CLR can detect many system-level errors and raise an exception if necessary. A managed application can endeavor to catch these exceptions and either attempt some form of recovery or at least fail in a graceful manner, reporting the reason for the exception and providing information that can help a developer to understand the cause of the exception in order to take steps to rectify the situation in the future.

CLR exceptions are specific to the .NET Framework. WCF is intended to build client applications and services that are interoperable with other environments. For example, a Java client application would not understand the format of a CLR exception raised by a WCF service or how to handle it. Part of the SOAP specification describes how to format and send errors in SOAP messages by using SOAP faults. The SOAP specification includes a schema for formatting SOAP faults as XML text and encapsulating them in a SOAP message. A SOAP fault must specify an error code and a text description of the fault (called the “reason”), and it can include other optional pieces of information. Interoperable services built using the WCF should convert .NET Framework exceptions into SOAP faults and follow the SOAP specification for reporting these faults to client applications.



More Info For a detailed description of the format and contents of a SOAP fault, see the World Wide Web Consortium Web site at <http://www.w3.org/TR/soap12-part1/#soapfault>.

Throwing and Catching a SOAP Fault

The WCF library provides the *FaultException* class in the *System.ServiceModel* namespace. If a WCF service throws a *FaultException* object, the WCF runtime generates a SOAP fault message that is sent back to the client application.

In the first set of exercises in this chapter, you will add code to the WCF *ProductsService* service that detects selected problems when accessing the *AdventureWorks* database and uses the *FaultException* class to report these issues back to the client application.

Modify the WCF Service to Throw SOAP Faults

1. Using Visual Studio, open the *ProductsServiceFault* solution located in the Microsoft Press\WCF Step By Step\Chapter 3\ProductsServiceFault folder (within your Documents folder).

This solution contains a copy of the *ProductsServiceLibrary*, *ProductsServiceHost*, and *ProductsClient* applications that you created in Chapter 2, “Hosting a WCF Service.”

2. In the `ProductsServiceLibrary` project, open the `ProductsService.cs` file to display the code for the service in the Code And Text Editor window.
3. Locate the `ListProducts` method in the `ProductsServiceImpl` class.

You should recall from Chapter 1, "Introducing Windows Communication Foundation," that this method uses the Entity Framework to connect to the `AdventureWorks` database and retrieve the product number of every product in the `Product` table. The product numbers are stored in a list which is returned to the client application. Notice that the exception handler for this method currently ignores all exceptions.

4. Modify the exception handler, as shown in bold in the following:

```
public List<string> ListProducts()
{
    ...
    try
    {
        ...
    }
    catch (Exception e)
    {
        // Edit the Initial Catalog in the connection string in app.config
        // to trigger this exception
        if (e.InnerException is System.Data.SqlClient.SqlException)
        {
            throw new FaultException(
                "Exception accessing database: " +
                e.InnerException.Message, new FaultCode("Connect to database"));
        }
        else
        {
            throw new FaultException(
                "Exception reading product numbers: " +
                e.Message, new FaultCode("Iterate through products"));
        }
    }

    // Return the list of product numbers
    return productList;
}
```

If an exception occurs, this code examines the cause. If the `InnerException` property of the `Exception` object is a `SqlException`, then the exception was caused by the code that accesses the database in the Entity Framework. If the exception is some other type, then the problem must lie in the code that iterates through the list of products retrieved from the database. In both cases, this code creates a new `System.ServiceModel.FaultException` object with the details of the exception and throws it. The operation will stop running and will instead generate a SOAP fault containing a description of the fault and a fault code (which for the purposes in this example simply specifies a name for identification). This SOAP fault is sent back to the client.



Note If you don't create a *FaultCode* object, the WCF runtime will itself automatically generate a *FaultCode* object named "Client" and add it to the SOAP fault sent back to the client.

5. Build the solution.

Modify the WCF Client Application to Catch SOAP Faults

1. In the ProductsClient project, open the file Program.cs to display the code for the client application in the Code And Text Editor window.
2. In the *Main* method, add a *try/catch* block around the code that calls the operations in the WCF service, as shown in bold in the following:

```
static void Main(string[] args)
{
    ...
    // Test the operations in the service

    try
    {
        // Obtain a list of all products
        ...

        // Fetch the details for a specific product
        ...

        // Query the stock level of this product
        ...

        // Modify the stock level of this product
        ...

        // Disconnect from the service
        ...
    }
    catch (FaultException e)
    {
        Console.WriteLine("{0}: {1}", e.Code.Name, e.Reason);
    }

    Console.WriteLine("Press ENTER to finish");
    Console.ReadLine();
}
```

If any of the operations generate a SOAP fault, the WCF runtime on the client creates a *FaultException* object. The catch handler for the *FaultException* object displays the fault code and reason. The name of the fault code is the value specified by the *FaultCode* constructor in the service, and the *Reason* string is the text description of the fault provided by the service.

Test the FaultException Handler



Important Before you perform this exercise, make sure that you still have port 8000 reserved for your application, as described in the exercise “Reserve HTTP Port 8000” in Chapter 2. To reserve this port, open a command prompt as Administrator, and run the following command (replace *UserName* with your Windows user name):

```
netsh http add urlacl url=http://+:8000/ user=Us er Name
```

1. In the ProductsServiceHost project, edit the App.config file. The `<connectionStrings>` section of this file contains the information used by the Entity Framework to connect to the *AdventureWorks* database.
2. In the `<add>` element of the `<connectionStrings>` section, change the *Initial Catalog* part of the *connectionString* attribute to refer to the **Junk** database, as follows (do not change any other parts of the *connectString* attribute):

```
<connectionStrings>
  <add ... connectionString="...;Initial Catalog=Junk;..." />
</connectionStrings>
```

3. Build and run the solution without debugging.

The Products Service Host window and the ProductsClient console window should both start.

4. In the Products Service Host window, click Start.

If a Windows Security Alert message box appears, click Allow Access.

5. When the service status in the Products Service Host window displays the message “Service Running,” press Enter in the ProductsClient console window.

After a short delay, the ProductsClient application reports an exception similar to the following when performing Test 1 (your message might vary if you are attempting to connect to the database as a different user):

```
C:\Windows\system32\cmd.exe
Press ENTER when the service has started
Test 1: List all products
Connect to database: Exception accessing database: Cannot open database "Junk" requested by the login. The login failed.
Login failed for user 'LON-DEU-01\John'.
Press ENTER to finish
```

The ProductsService service failed when attempting to connect to the database—the SOAP fault code is “Connect to database.”

6. Press Enter to close the ProductsClient console.
7. Click Stop in the Products Service Host window, and then close the application.
8. In the App.config file for the ProductsServiceHost application, change the database back to **AdventureWorks** in the `<connectionString>` attribute.
9. In the `ListProducts` method in the ProductsService.cs file, comment out the code that instantiates the `productsList` object and replace it with code that sets this object to **null**. In the body of the `try` block, add a statement that clears the `productsList` object before assigning it the data retrieved from the database, as shown in bold in the following:

```
public List<string> ListProducts()
{
    // Create a list for holding product numbers
    List<string> productsList = null; // new List<string>();

    try
    {
        // Fetch the product number of every product in the database
        var products = from product in database.Products
                       select product.ProductNumber;

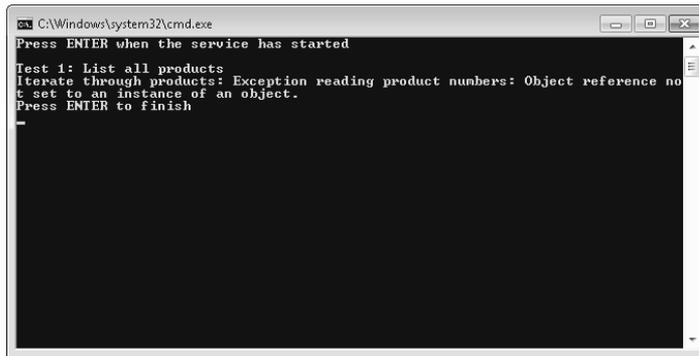
        productsList.Clear();
        productsList = products.ToList();
    }
    catch (Exception e)
    {
        ...
    }
    ...
}
```



Note The statement that calls `Clear` is actually redundant and is only used by this exercise to illustrate generating an exception that results in a SOAP fault.

10. Build and run the solution again, without debugging.
11. In the Products Service Host window, click Start.
12. When the service is running press Enter in the ProductsClient console window.

The ProductsClient application reports a different exception when performing Test 1:



```
C:\Windows\system32\cmd.exe
Press ENTER when the service has started

Test 1: List all products
Iterate through products: Exception reading product numbers: Object reference not
set to an instance of an object.
Press ENTER to finish
```

This time, the *ProductsService* service failed when clearing the list of products prior to iterating through the items retrieved from the database and adding them to the *productsList* collection (which is now set to *null*)—the SOAP fault code is “Iterate through products”; the reason explains that an object reference was not initialized correctly.

13. Press Enter to close the ProductsClient console window.
14. Click Stop in the Products Service Host form, and then close the application.



Note Do not change the code in the *Listproducts* method back to the correct version just yet.

Using Strongly Typed Faults

Throwing a *FaultException* is very simple but is actually not as useful as it first appears. A client application must examine the *FaultException* object that it catches to determine the cause of the error, so it is not easy to predict what possible exceptions could occur when invoking a WCF service. In such situations, all developers can do is write generalized catch handlers with very limited scope for recovering from specific exceptions. You can think of this as analogous to using the *System.Exception* type to throw and handle exceptions in regular .NET Framework applications. A better solution is to use strongly typed SOAP faults.

In Chapter 1, you saw that a service contract for a WCF service contains a series of operation contracts defining the methods, or operations, that the service implements. A service contract can additionally include information about any faults that might occur when executing an operation. If an operation in a WCF service detects an exception, it can generate a specific SOAP fault message that it can send back to the client application. The SOAP fault message should contain sufficient detail so the user or an administrator can understand the reason for the exception and, if possible, take any necessary corrective action. A client application can use the fault information in the service contract to anticipate faults and provide specific handlers that can catch and process each different fault. These are strongly typed faults.

You specify the possible faults that can occur by using the *FaultContract* attribute in a service contract. This is what you will do in the next set of exercises.



Note You can only apply the *FaultContract* attribute to operations that return a response. You cannot use them with one-way operations. You will learn more about one-way operations in Chapter 12, “Implementing One-Way and Asynchronous Operations.”

Use the FaultContract Attribute to Specify the SOAP Faults an Operation Can Throw

1. In the ProductsServiceFault solution, in the ProductsServiceLibrary project, open the IProductsService.cs file.
2. In the IProductsService.cs file, add the following classes shown in bold to the *Products* namespace:

```
namespace Products
{
    // Classes for passing fault information back to client applications
    [DataContract]
    public class SystemFault
    {
        [DataMember]
        public string SystemOperation { get; set; }

        [DataMember]
        public string SystemReason { get; set; }

        [DataMember]
        public string SystemMessage { get; set; }
    }

    [DataContract]
    public class DatabaseFault
    {
        [DataMember]
        public string DbOperation { get; set; }

        [DataMember]
        public string DbReason { get; set; }

        [DataMember]
        public string DbMessage { get; set; }
    }

    // Data contract describing the details of a product
    ...

    // Service contract describing the operations provided by the WCF service
    ...
}
```

These classes define types that you will use for passing the details of SOAP faults as exceptions from a service back to a client. Note that although both classes have a similar shape, you can pass almost any type of information in a SOAP fault; the key point is that the type and its members must be serializable. These two classes use the *DataContract* and *DataMember* attributes to specify how they should be serialized.

3. Locate the *IProductsService* interface at the end of the *IProductsService.cs* file.

Remember that this interface defines the service contract for the *ProductsService*.

4. In the *IProductsService* interface, modify the definition of the *ListProducts* operation, as shown in bold in the following code:

```
// Service contract describing the operations provided by the WCF service
[ServiceContract]
public interface IProductsService
{
    // Get the product number of every product
    [FaultContract(typeof(SystemFault))]
    [FaultContract(typeof(DatabaseFault))]
    [OperationContract]
    List<string> ListProducts();

    // Get the details of a single product
    ...

    // Get the current stock level for a product
    ...

    // Change the stock level for a product
    ...
}
```

The *FaultContract* attributes indicate that the *ListProducts* method can generate SOAP faults, which a client application should be prepared to handle. The parameter to the *FaultContract* attribute specifies the information that the SOAP fault will contain. In this case, the *ListProducts* operation can generate two types of SOAP faults: one based on the *SystemFault* type, and the other based on the *DatabaseFault* type.

Modify the WCF Service to Throw Strongly Typed Faults

1. In the *ProductsService.cs* file, locate the *ListProducts* method in the *ProductsServiceImpl* class.
2. Replace the code in the *catch* block that traps *SqlException* exceptions, as shown in bold in the following:

```

public List<string> ListProducts()
{
    ...
    try
    {
        ...
    }
    catch (Exception e)
    {
        // Edit the Initial Catalog in the connection string in app.config
        // to trigger this exception
        if (e.InnerException is System.Data.SqlClient.SqlException)
        {
            DatabaseFault dbf = new DatabaseFault
            {
                DbOperation = "Connect to database",
                DbReason = "Exception accessing database",
                DbMessage = e.InnerException.Message
            };

            throw new FaultException<DatabaseFault>(dbf);
        }
        else
        {
            ...
        }
    }
    ...
}

```

This block creates and populates a *DatabaseFault* object with the details of the exception. The *throw* statement creates a new *FaultException* object based on this *DatabaseFault* object. Note that in this case, the code makes use of the generic *FaultException* class; the type parameter specifies a serializable type with the type-specific details of the exception. At runtime, WCF uses the information in this object to create a SOAP fault message. The *FaultException* constructor is overloaded, and you can optionally specify a reason message and a fault code as well as the *DatabaseFault* object.

3. Replace the code in the *else* part of the *catch* block with that shown in bold, as follows:

```

public List<string> ListProducts()
{
    ...
    try
    {
        ...
    }
    catch (Exception e)
    {
        // Edit the Initial Catalog in the connection string in app.config
        // to trigger this exception
        if (e.InnerException is System.Data.SqlClient.SqlException)

```

```

    {
        ...
    }
    else
    {
        SystemFault sf = new SystemFault
        {
            SystemOperation = "Iterate through products",
            SystemReason = "Exception reading product numbers",
            SystemMessage = e.Message
        };

        throw new FaultException<SystemFault>(sf);
    }
    ...
}

```

This block of code is similar to the previous *catch* code, except that it creates a *System Fault* object and throws a *FaultException* based on this object. The rationale behind using a different type for the exception is that the kinds of exceptions that could arise when accessing a database are fundamentally different from the exceptions that could occur when reading configuration information. Although not shown in this example, the information returned by a database access exception could be quite different from the information returned by a system exception.

4. Build the solution.

You can now modify the client application to handle the exceptions thrown by the service. However, first you must regenerate the proxy class that the client uses to communicate with the service. The service is not currently running, so you cannot use the *Update Service Reference* feature of Visual Studio. Instead, you will use the *svcutil* utility to generate the proxy class from the assembly containing the *ProductsService* service.

Regenerate the Proxy Class for the WCF Client Application

1. Open a Visual Studio command prompt window and move to the folder `\Microsoft Press\WCF Step By Step\Chapter 3\ProductsServiceFault\ProductsServiceLibrary\bin\Debug` folder.
2. Run the following command:

```
svcutil ProductsServiceLibrary.dll
```

This command runs the *svcutil* utility to extract the definition of the *ProductsService* service and the other types from the *ProductsServiceLibrary* assembly. It generates the following files:

- **Products.xsd** This is an XML schema file that describe the structure of the, *DatabaseFault*, *SystemFault*, and *ProductData* types. The *svcutil* utility uses the information specified in the data contracts for these types to generate this file. Part of this file, displaying the *DatabaseFault* type, is shown in the following:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:tns="http://schemas.datacontract.org/2004/07/Products"
  elementFormDefault="qualified"
  targetNamespace="http://schemas.datacontract.org/2004/07/Products"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="DatabaseFault">
    <xs:sequence>
      <xs:element minOccurs="0" name="DbMessage" nillable="true"
        type="xs:string" />
      <xs:element minOccurs="0" name="DbOperation" nillable="true"
        type="xs:string" />
      <xs:element minOccurs="0" name="DbReason" nillable="true"
        type="xs:string" />
    </xs:sequence>
  </xs:complexType>
  <xs:element name="DatabaseFault" nillable="true" type="tns:DatabaseFault" />
  ...
</xs:schema>
```

- **Tempuri.org.xsd** This is another XML schema file. This schema describes the messages that a client can send to, or receive from, the *ProductsService* service. You will see later (in the WSDL file for the service) that each operation in the service is defined by a pair of messages; the first message in the pair specifies the message that the client must send to invoke the operation, and the second message specifies the response sent back by the service. This file references the data contract in the *Products.xsd* file to obtain the description of the *ProductData* type used by the response message of the *GetProduct* operation. The portion of this file that defines the messages for the *ListProducts* and *GetProduct* operations appears as follows:

```
<?xml version="1.0" encoding="utf-8"?>
...
<xs:element name="ListProducts">
  <xs:complexType>
    <xs:sequence />
  </xs:complexType>
</xs:element>
<xs:element name="ListProductsResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="ListProductsResult" nillable="true"
        xmlns:q1="http://schemas.microsoft.com/2003/10/Serialization/Arrays"
        type="q1:ArrayOfstring" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```

<xs:element name="GetProduct">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="productNumber" nillable="true"
        type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="GetProductResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="GetProductResult" nillable="true"
        xmlns:q2="http://schemas.datacontract.org/2004/07/Products"
        type="q2:ProductData" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
...
</xs:schema>

```



Note The name of this file and the namespace of the types in this file are dictated by the *ServiceContract* attribute of the interface implemented by the service. The name *Tempuri.org* is the default namespace. You can change it by specifying the *Namespace* parameter in the *ServiceContract* attribute, like this:

```
[ServiceContract (Namespace="Adventure-Works.com")]
```

- ❑ **Schemas.microsoft.com.2003.10.Serialization.Arrays.xsd** This file is another XML schema that describes how to represent an array of strings in a SOAP message. The *ListProducts* operation references this information in the *ListProducts Response* message. The value returned by the *ListProducts* operation is a list of strings containing product numbers. As described in Chapter 1, the .NET Framework generic *List<>* type is serialized as an array when transmitted as part of a SOAP message.
- ❑ **Schemas.microsoft.com.2003.10.Serialization.xsd** This XML schema file describes how to represent the primitive types (such as float, int, decimal, and string) in a SOAP message, as well as some other built-in types frequently used when sending SOAP messages.
- ❑ **Tempuri.org.wsdl** This file contains the WSDL description of the service, describing how the messages and data contracts are used to implement the operations that a client application can invoke. It references the XML schema files to define the data and messages that implement operations. Notice that the definition of the *ListProducts* operation includes the two fault messages that you defined earlier:

```

...
<wsdl:operation name="ListProducts">
  <soap:operation soapAction="http://tempuri.org/IProductsService/ListProducts"
style="document" />
  <wsdl:input>
    <soap:body use="literal" />
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal" />
  </wsdl:output>
  <wsdl:fault name="DatabaseFaultFault">
    <soap:fault name="DatabaseFaultFault" use="literal" />
  </wsdl:fault>
  <wsdl:fault name="SystemFaultFault">
    <soap:fault name="SystemFaultFault" use="literal" />
  </wsdl:fault>
</wsdl:operation>

```

You can use the WSDL file and the XML schema files to generate the proxy class.

3. In the Visual Studio command prompt window, type the following command:

```
svcutil /namespace:*,ProductsClient.ProductsService tempuri.org.wsdl *.xsd
```



Note The character between the asterisk (*) and the string *ProductsClient.ProductsService* is a comma (,).

This command runs the *svcutil* utility again, but this time it uses the information in the WSDL file and all the schema files (*.xsd) to generate a C# source file containing a class that can act as a proxy object for the service. The namespace parameter specifies the C# namespace generated for the class (the namespace shown here has been selected to be the same as that generated by Visual Studio in the exercises in Chapter 1, to minimize the changes required to the code in the client application; however, you will need to modify the client configuration file to match this namespace). The *svcutil* utility creates two files:

- **Products.cs** This is the source code for the proxy class.
- **Output.config** This is an example application configuration file that the client application could use to configure the proxy to communicate with the service. By default, the configuration file generates an endpoint definition with the *basicHttpBinding* binding.



Note You can also use the *svcutil* utility to generate a proxy directly from a Web service endpoint rather than generating the metadata from an assembly. This is what Visual Studio does when you use the *Add Service Reference* feature. For more information about the *svcutil* utility, see the "ServiceModel Metadata Utility Tool" on the Microsoft Web site at <http://msdn.microsoft.com/en-us/library/aa347733.aspx>.

4. In Visual Studio, in the ProductsClient project, copy the app.config file and paste the copied file back into the ProductsClient project with the default name, Copy of app.config. This step is necessary because the next step will remove some important information from the app.config file that you will need later.
5. In the ProductsClient project, delete the *ProductsService* service from the Service References folder. As well as removing the service reference, this action also deletes the configuration information for accessing the service from the app.config file, which is why you made a copy of the original version of this file in the previous step.
6. Add the file Products.cs that you have just created to the ProductsClient project. This file is located in the Microsoft Press\WCF Step By Step\Chapter 3\ProductsServiceFault\ProductsServiceLibrary\bin\Debug folder.
7. Delete the app.config file from the ProductsClient project and rename the file Copy of app.config as **app.config**.
8. Open the app.config file in the Code And Text Editor window. Change the contract for both client endpoints to **ProductsClient.ProductsService.IProductsService**, as shown in bold in the following.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <bindings>
      ...
    </bindings>
    <client>
      <endpoint address="http://localhost:8000/ProductsService/Service.svc"
        binding="basicHttpBinding"
        bindingConfiguration="BasicHttpBinding_IProductsService"
        contract="ProductsClient.ProductsService.IProductsService"
        name="BasicHttpBinding_IProductsService" />
      <endpoint address="net.tcp://localhost:8080/TcpService" binding="netTcpBinding"
        contract="ProductsClient.ProductsService.IProductsService"
        name="NetTcpBinding_IProductsService" />
    </client>
  </system.serviceModel>
</configuration>
```

This change is necessary as you generated the types for the proxy in the *ProductsClient*. *ProductsService* namespace when you ran the *svcutil* utility.



Note You could have copied the output.config file generated by the *svcutil* utility to the ProductsClient project and renamed it as app.config rather than preserving and editing the original app.config file. However, although the output.config file specifies the correct type name for the contract attribute of the endpoint, it does not include the address of the service, so you would have had to edit the file and add this information. Additionally, the output.config file only contains the definition of a single *BasicHttpBinding* endpoint, so you would also have needed to add the definition of the *NetTcpBinding* endpoint. It was simpler to modify the existing app.config file!

Modify the WCF Client Application to Catch Strongly Typed Faults

1. In the ProductsClient project, open the Program.cs file in the Code And Text Editor window.
2. Add the following *catch* handlers shown in bold after the *try* block in the *Main* method (leave the existing *FaultException* handler in place as well):

```
static void Main(string[] args)
{
    ...
    try
    {
        ...
    }
    catch (FaultException<SystemFault> sf)
    {
        Console.WriteLine("SystemFault {0}: {1}\n{2}",
            sf.Detail.SystemOperation, sf.Detail.SystemMessage,
            sf.Detail.SystemReason);
    }
    catch (FaultException<DatabaseFault> dbf)
    {
        Console.WriteLine("DatabaseFault {0}: {1}\n{2}",
            dbf.Detail.DbOperation, dbf.Detail.DbMessage,
            dbf.Detail.DbReason);
    }
    catch (FaultException e)
    {
        Console.WriteLine("{0}: {1}", e.Code.Name, e.Reason);
    }
    ...
}
```

These two handlers catch the *SystemFault* and *DatabaseFault* faults. Notice that the fields containing the exception information that are populated by the *ProductsService* (*SystemOperation*, *SystemMessage*, *SystemReason*, *DbOperation*, *DbMessage*, and *DbReason*) are located in the *Detail* field of the *FaultException* object.



Important You must place these two exception handlers before the non-generic *FaultException* handler. The non-generic handler would attempt to catch these exceptions if it occurred first, and the compiler would not let you build the solution.

3. Build and run the solution without debugging.
4. When the Products Service Host window appears, click Start to run the service.

5. When the service has started, in the client application console window, press Enter.

The code in the *ListProducts* method in the *ProductsService* service still generates a null reference exception. The service throws a *FaultException*, containing a *SystemFault* object, which is serialized as a SOAP fault. The client application catches this fault and displays the details.

6. Press Enter to close the client application. Stop the service and close the Products Service Host window.
7. Edit the *ProductsService.cs* file in the *ProductsServiceLibrary* project. In the *ListProducts* method, restore the statement that initializes the *productsList* variable back to its original state and remove the code in the *try* block that calls the *Clear* method, as shown in the following:

```
public List<string> ListProducts()
{
    // Create a list for holding product numbers
    List<string> productsList = new List<string>();

    try
    {
        // Connect to the AdventureWorks database by using the Entity Framework
        using (AdventureWorksEntities database = new AdventureWorksEntities())
        {
            // Fetch the product number of every product in the database
            var products = from product in database.Products
                           select product.ProductNumber;

            productsList = products.ToList();
        }
    }
    catch (Exception e)
    {
        ...
    }
    ...
}
```

8. Edit the *App.config* file in the *ProductsServiceHost* project by using the Code And Text Editor window, and change the *Initial Catalog* part of the *connectionString* attribute to refer to the **Junk** database, as you did earlier:

```
<connectionStrings>
  <add ... connectionString="...;Initial Catalog=Junk;..." />
</connectionStrings>
```

9. Build and run the solution without debugging.
10. When the Products Service Host window appears, click Start to run the service.

11. When the service has started, in the client application console window, press Enter.

The application configuration file for the service host application again refers to an invalid database. This “mistake” causes the service to generate a SOAP fault containing a *DatabaseFault* with details of the failure. The ProductsClient application catches this exception in the *FaultException<DatabaseFault>* handler.

12. Press Enter to close the client application. Stop the service and close the Products Service Host window.
13. Correct the *Initial Catalog* attribute in the app.config file for the ProductsServiceHost project and set it back to refer to the *AdventureWorks* database, as follows:

```
<connectionStrings>
  <add ... connectionString="...;Initial Catalog=AdventureWorks;..." />
</connectionStrings>
```

14. Build and run the solution without debugging.
15. In the Products Service Host window, start the service. Press Enter in the client application console window. Verify that the code now runs without any exceptions. Close the client console window, stop the service, and close the Products Service Host window when you have finished.

Reporting Unanticipated Exceptions

Specifying the possible exceptions that a service can throw when performing an operation is an important part of the contract for a service. If you use strongly-typed exceptions, you must specify every exception that an operation can throw in the service contract. If a service throws a strongly-typed exception that is not specified in the service contract, the details of the exception are not propagated to the client—the exception does not form part of the WSDL description of the operation used to generate the client proxy. There will inevitably be situations where it is difficult to anticipate the exceptions that an operation could throw. In these cases, you should catch the exception in the service, and if you need to send it to the client, raise an ordinary (non-generic) *FaultException* as you did in the first set of exercises in this chapter.

While you are developing a WCF service, it can be useful to send information about all exceptions that occur in the service—anticipated or not—to the client application for debugging purposes. You will see how you can achieve this in the next set of exercises.

Modify the WCF Service to Throw an Unanticipated Exception

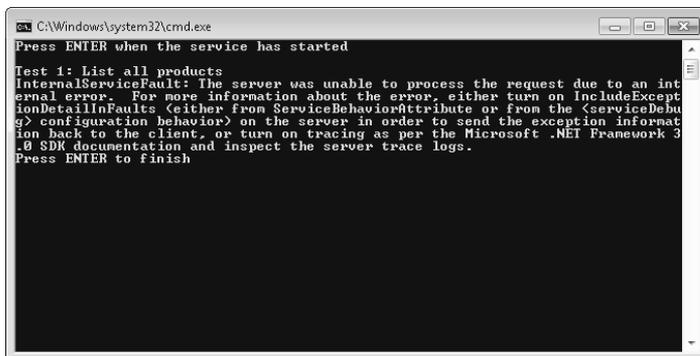
1. In the ProductsServiceFault solution, in the ProductsServiceLibrary project, edit the ProductsService.cs file.
2. Add the following statement (shown in bold) as the first line of code in the *ListProducts* method in the *IProductsImpl* class:

```
public List<string> ListProducts()  
{  
    int i = 0, j = 0, k = i / j;  
    ...  
}
```

This statement will generate a *DivideByZeroException*. Note that the method does not trap this exception, and it is not mentioned in the service contract.

3. Build and run the solution without debugging.
4. In the Products Service Host window, click Start. In the client application console window, press Enter to connect to the service and invoke the *ListProducts* operation.

The service throws the *DivideByZero* exception. However, the details of the exception are not forwarded to the client application. Instead, the WCF runtime generates a very nondescript SOAP fault that is caught by the *DefaultException* handler in the client:



```
C:\Windows\system32\cmd.exe  
Press ENTER when the service has started  
Test 1: List all products  
InternalServiceFault: The server was unable to process the request due to an internal error. For more information about the error, either turn on IncludeExceptionDetailInFaults (either from ServiceBehaviorAttribute or from the <serviceDebug> configuration behavior) on the server in order to send the exception information back to the client, or turn on tracing as per the Microsoft .NET Framework 3.0 SDK documentation and inspect the server trace logs.  
Press ENTER to finish
```

This lack of detail is actually a security feature. If the service provided a complete description of the exception to the client, then, depending on the information provided, a malicious user could glean potentially useful information about the structure of the service and its internal workings.

5. Close the client console window. Stop the service and close the Products Service Host window.

In the next exercise you will configure the host server to provide detailed information about unanticipated exceptions.

Configure the WCF Service to Send Details of Exceptions

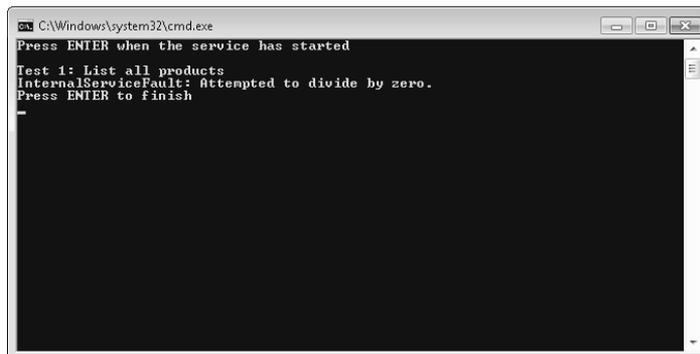
1. In the ProductsServiceHost project, edit the App.config file by using the Code And Text Editor window.
2. In the `<serviceBehaviors>` section, edit the `<serviceDebug>` element in the `<behavior>` section and set the `includeExceptionDetailInFaults` attribute to true:

```
<?xml version="1.0"?>
<configuration>
  ...
  <system.serviceModel>
    ...
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <!-- To avoid disclosing metadata information, set the value below to
               false and remove the metadata endpoint above before deployment -->
          <serviceMetadata httpGetEnabled="false"/>
          <!-- To receive exception details in faults for debugging purposes, set
               the value below to true. Set to false before deployment to avoid
               disclosing exception information -->
          <serviceDebug includeExceptionDetailInFaults="true"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
    ...
  </system.serviceModel>
</configuration>
```

Setting the `includeExceptionDetailInFaults` attribute to `true` causes WCF to transmit the full details of exceptions when it generates SOAP faults for unanticipated errors.

3. Build and run the solution with debugging.
4. In the Products Service Host window, click Start. In the client application console window, press Enter.

The service throws the `DivideByZero` exception. This time, the client is sent specific information about the exception and reports it:



```
C:\Windows\system32\cmd.exe
Press ENTER when the service has started
Test 1: List all products
InternalServiceFault: Attempted to divide by zero.
Press ENTER to finish
```

5. Close the client console window. Stop the service and close the Products Service Host window.
6. In the ProductsServiceLibrary project, edit the ProductsService.cs file.
7. In the *ListProducts* method, comment out the line of code that causes the *DivideByZeroException* exception.
8. In the App.config file for the ProductsServiceHost project, set the *includeExceptionDetailInFaults* attribute of the `<serviceDebug>` element to false.
9. Build and run the solution without debugging.
10. In the Products Service Host window, start the service. Press Enter in the client application console window. Verify that the code runs without any exceptions. Close the client console window and the Products Service Host window when you have finished.

The previous exercise used the application configuration file to specify the *serviceDebug* behavior for the service. You can perform the same task by using the *ServiceBehavior* attribute of the class that implements the service, like this:

```
[ServiceBehavior(IncludeExceptionDetailInFaults=true)]
public class ProductsServiceImpl : IProductsService
{
    ...
}
```

However, it is recommended that you enable this behavior only by using the application configuration file. There are a couple of good reasons for this:

- You can turn the behavior on and off in the configuration file without rebuilding the application. You should not deploy an application to a production environment with this behavior enabled, and it is very easy to forget that you have enabled this behavior if you use the *ServiceBehavior* attribute in code.
- If you enable this behavior in code, you *cannot* disable it by altering the application configuration file. Rather more confusingly, if you *disable* this behavior in code, you *can* enable it in the application configuration file. The general rule is that if the *IncludeExceptionDetailInFaults* behavior is enabled either in code or in the application configuration file, it will work. It must be disabled in *both* places to turn it off. Keep life simple by only specifying this behavior in one place—the application configuration file.

Managing Exceptions in Service Host Applications

In Chapter 2, you saw how to create a host application for a WCF service and use this application to control the lifecycle of the service. A service host application uses a *ServiceHost* object to instantiate and manage a WCF service. The *ServiceHost* class implements a finite-state machine. A *ServiceHost* object can be in one of a small number of states, and there are well-defined rules that determine how the WCF runtime transitions a *ServiceHost* object from one state to another. Some of these transitions occur as the result of specific method calls, while others are caused by exceptions in the service, in the communications infrastructure, or in the objects implementing the channel stack. A service host application should be prepared to handle these transitions and attempt recovery to ensure that the service is available whenever possible.

ServiceHost States and Transitions

When you instantiate a *ServiceHost* object, it starts in the *Created* state. In this state, you can configure the object; for example, you can use the *AddServiceEndpoint* method to cause the *ServiceHost* object to listen for requests on a particular endpoint. A *ServiceHost* object in this state is not ready to accept requests from client applications.

You start a *ServiceHost* object listening for requests by using the *Open* method (or the *BeginOpen* method if you are using the asynchronous programming model). The *ServiceHost* object moves to the *Opening* state while it creates the channel stacks specified by the bindings for each endpoint and starts the service. If an exception occurs at this point, the object transitions to the *Faulted* state. If the *ServiceHost* object successfully opens the communication channels for the service, it moves to the *Opened* state. Only in this state can the object accept requests from client applications and direct them to the service.

You stop a *ServiceHost* object from listening for client requests by using the *Close* (or *BeginClose*) method. The *ServiceHost* object enters the *Closing* state. Currently running requests are allowed to complete, but clients can no longer send new requests to the service. When all outstanding requests have finished, the *ServiceHost* object moves to the *Closed* state. You can also stop a service by using the *Abort* method. This method closes the service immediately without waiting for the service to finish processing client requests. *Stopping* or aborting the service disposes the service object hosted by the *ServiceHost* object and reclaims any resources it was using. To start the service, you must recreate the *ServiceHost* object with a new instance of the service and then execute the *Open* method to reconstruct the channel stacks and start listening for requests again.

A *ServiceHost* object enters the *Faulted* state either when it fails to open correctly or if it detects an unrecoverable error in a channel used by the *ServiceHost* object to communicate with clients (for example, if some sort of protocol error occurs). When a *ServiceHost* object is in the *Faulted* state, you can examine the properties of the object to try and ascertain the cause of the failure, but you cannot send requests to the service. To recover the service, you should use the *Abort* method to close the service, recreate the *ServiceHost* object, and then execute the *Open* method again. Figure 3-1 summarizes the state transitions for a *ServiceHost* object along with the methods and conditions that cause the object to move between states.



Tip You can determine the current state of a *ServiceHost* object by examining the value of the *State* property.

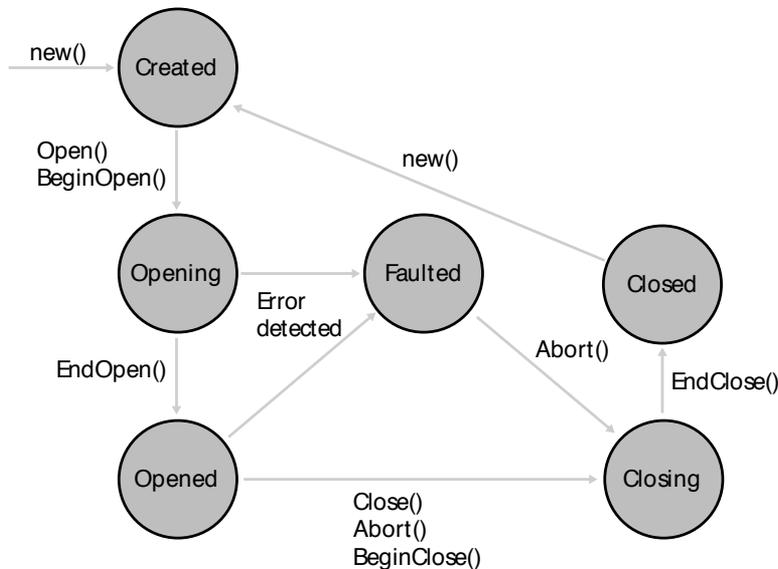


FIGURE 3-1 State transition diagram for the *ServiceHost* class.

Handling Faults in a Host Application

When a *ServiceHost* object moves from one state to another, it can trigger an event. These events were described in Table 2-2 in Chapter 2. From an error-handling perspective, the most important of these is the *Faulted* event, which occurs when a *ServiceHost* object enters the *Faulted* state. You should subscribe to this event, and provide a method that attempts to determine the cause, and then abort and restart the service, like this:

```
// ServiceHost object for hosting a WCF service
ServiceHost productsServiceHost;
productsServiceHost = new ServiceHost(...);
...
// Subscribe to the Faulted event of the productsServiceHost object
productsServiceHost.Faulted += (eventSender, eventArgs) =>
{
    // FaultHandler method
    // Runs when productsServiceHost enters the Faulted state

    // Examine the properties of the productsServiceHost object
    // and log the reasons for the fault
    ...
    // Abort the service
    productsServiceHost.Abort();

    // Recreate the ServiceHost object
    productsServiceHost = new ServiceHost(...);

    // Start the service
    productsServiceHost.Open();
};
...
```



Note You can use the *Close* method rather than *Abort* in the fault handler, but a service in the faulted state will not be able to continue processing current requests or receive new ones. Using the *Abort* method to close the service can reduce the time required in the *FaultHandler* method to restart the service.

Handling Unexpected Messages in a Host Application

One other exceptional circumstance that can arise in a host application is an unexpected message from a client. Client applications built by using the WCF library typically communicate with the service by using a proxy object, generated by using the *svcutil* utility. The proxy object provides a strongly-typed interface to the service that specifies the operations the client can request (and therefore the messages that the client sends). It is unlikely that a WCF client using a correctly generated proxy object will send an unexpected message. However, remember that a WCF service is simply a service that accepts SOAP messages, and developers building client applications can use whatever means they see fit for sending these messages. Developers building Java client applications will typically use Java-specific tools and libraries for constructing and sending SOAP messages. WCF also provides a low-level mechanism that allows developers to open a channel to a service, create SOAP messages, and then send them to the service, as shown in this fragment of code:

```
// Create a binding and endpoint to communicate with the ProductsService
BasicHttpBinding binding = new BasicHttpBinding();

EndpointAddress address = new EndpointAddress(
    "http://localhost:8000/ProductsService/Service.svc");

ChannelFactory<IRequestChannel> factory = new
    ChannelFactory<IRequestChannel>(binding, address);

// Connect to the ProductsService service
IRequestChannel channel = factory.CreateChannel();
channel.Open();

// Send a ListProducts request to the service
Message request = Message.CreateMessage(MessageVersion.Soap11,
    "http://tempuri.org/IProductsService/ListProducts");
Message reply = channel.Request(request);

// Process the reply
// (should be a SOAP message with a list of product numbers)
...
// Release resources and close the connection
reply.Close();
channel.Close();
factory.Close();
```

Don't worry too much about the details of this block of code—you will learn more about using `Message` and `Channel` objects in Chapter 11, “Programmatically Controlling the Configuration and Communications.” The key statement is the line that creates the message sent to the `ProductsService` service:

```
Message request = Message.CreateMessage(MessageVersion.Soap11,
    "http://tempuri.org/IProductsService/ListProducts");
```

The second parameter to the `CreateMessage` method specifies the action that identifies the message sent to the service. If you recall the earlier discussion in this chapter describing the use of the `svcutil` utility to generate the client proxy, one of the files generated contained the WSDL description of the service. The WSDL description includes the definitions of each of the operations exposed by the service and the messages that an application sends to invoke these operations. Here is part of the WSDL describing the `ListProducts` operation:

```
<wsdl:operation name="ListProducts">
  <wsdl:input wsaw:Action="http://tempuri.org/IProductsService/ListProducts"
message="tns:IProductsService_ListProducts_InputMessage" />
  ...
</wsdl:operation>
```

When the service receives a message identified by the action *http://tempuri.org/IProducts/Service/ListProducts*, it performs the *ListProducts* operation. If a client application sends a message specifying an action that the service does not recognize, the service host application raises the *UnknownMessageReceived* event. The host application can catch this event and record the unrecognized message, like this:

```
// ServiceHost object for hosting a WCF service
ServiceHost productsServiceHost;
productsServiceHost = new ServiceHost(...);
...
// Subscribe to the UnknownMessageReceived event of the
// productsServiceHost object
productsServiceHost.UnknownMessageReceived += (eventSender, eventArgs) =>
{
    // UnknownMessageReceived event handler

    // Log the unknown message
    ...
    // Display a message to the administrator
    MessageBox.Show(string.Format(
        "A client attempted to send the message: {0} ",
        eventArgs.Message.Headers.Action));
};
...
```

There could be a perfectly innocent explanation for a client sending a message such as this, or it could be part of a more concerted attack by a malicious user trying to probe a service and gather information about the operations it supports.



Important The default value for the *httpGetEnabled* property of the *serviceMetadata* behavior is *false*, so unless you explicitly set it to *true*, WCF services do not publish their metadata. It is also worth noting that if you create a WCF service by using Visual Studio, the WCF Service template sets *httpGetEnabled* to *true*. Unless you explicitly need client applications to be able to access the metadata of a service, you should make sure that you reset this property to **false** when you deploy the service to a production environment.

One other possibility is that a WCF client application is using an out-of-date proxy object for sending messages to the service. If a developer modifies the service contract for a WCF service, she might change the messages that the service sends and receives. If any client applications that use the service are not updated, they might send messages that the service no longer understands. Therefore, if you update a service, you should ensure that you retain backward compatibility with existing clients. The same issues can arise with data contracts. You will learn more about how to update data contracts for a WCF service safely in Chapter 6, "Maintaining Service Contracts and Data Contracts."

Summary

In this chapter, you have seen how to use the *FaultException* class to send information about exceptions back to client applications as SOAP faults. You have seen how to use the *Fault Contract* attribute to specify the faults that a service can send and how to catch these faults in a client application. You have also seen how to propagate information about unanticipated exceptions from a service to a client for debugging purposes. You should understand how to make a service host application robust by tracking the states of a service, recovering from faults, and handling unexpected messages sent by client applications.

This page intentionally left blank

Index

A

- aborting transaction 357
- Abort method of ServiceHost object 114
- "Access is denied" message 158
- access rights 583
- AcknowledgementRange element 386
- acknowledgment message
 - from WCF runtime 386
- <AckRequested> block in SOAP header 385
- ActionMessageFilter class 526
- ActionMessageFilter object 532, 544
 - for ChannelDispatcher object 527
- Action property for OperationContract attribute 428, 532
- activation for WCF service
 - configuring 171–174
- Active Directory 123
 - for storing user roles 153
- Active Directory Certificate Services 138
- Active Directory Federation Services 2.0 642
- activities
 - passing information from workflow into 303
 - returning value from Execute method of 303
- Activity Designer 307
- Add Application dialog box 196
- AddBindingParameters method 411, 412
- AddDefaultEndpoints method of ServiceHost class 82
- Adding Behavior Element Extension Sections dialog box 153, 179, 493
- Adding Binding Element Extension Sections dialog box 392, 394
- AddItemToCart activity 342
- AddItemToCart message 385
- AddItemToCart method 265, 357–358, 468
 - implementing 247–248
 - OperationBehavior attribute 354–355
- AddItemToCart.xaml file 341
- Add New Web Site dialog box 13
- address-based routing 523
- AddressFilter property of EndpointDispatcher object 526
- Add Service Reference dialog box 30, 441, 588, 666
- Add Service Reference Wizard 203, 314, 419, 446, 588
- Add Web Reference command 650
- ad hoc discovery 492–499
 - disadvantages 499–546
- administrator, running as Visual Studio 39
- ADO.NET Entity Framework 10
 - entity model 579
- Advanced Encryption Standard (AES) 128-bit algorithm 128
- Advanced Settings dialog box 51
- AdventureWorksAdminHost project 458
- AdventureWorks Administrative Operations service
 - adding asynchronous operation 448–452
 - CalculateTotalValueOfStock operation 447–457
 - client applications for testing 440
 - creating 436–440
- AdventureWorksAdmin queue, creating and testing 462–463
- AdventureWorksAdminTestClient project 458
- AdventureWorks database
 - Internet Explorer for data display from 585–587
 - script to query 368
- AdventureWorksEntities class 582
- AdventureWorks OLTP sample database 9–10
- AfterReceiveRequest method 409
- AJAX (Asynchronous JavaScript and XML) 5
- AlgorithmSuite property 145
- AllowedImpersonationLevel property 161–162
- announcementEndpoint endpoint 519
- Announcement Endpoints folder 501

announcement messages

- client application modification to capture 501
 - ProductsService service configuring to send to discovery proxy 519
- announcement requests, caching in dictionary collection 517
- announcements
- ProductsService service configuration to send 500–501
- AnnouncementService class 500
- Anonymous value for AllowedImpersonationLevel property 162
- App.config file 32
- <client> section 71
 - <connectionStrings> section 26, 97
 - contract for client endpoints 107
 - <endpoint> configuration section 41
 - for ShoppingCartHost project 418
 - for ProductsClient application 81
 - <protocolMapping> section 83
 - <serviceBehaviors> section 85, 112
 - <service> section 70
 - endpoint added 73–74
 - for ShoppingCartClient application 254
 - for ShoppingCartHost project 529
 - <system.serviceModel> section 252
- AppFabric. *See* Windows Server AppFabric
- appid parameter 140
- application configuration file. *See also* App.config file
- certificate details in 194
 - from Entity Data Model wizard 12
 - for service host application 110
- application-defined endpoints, or system-defined 518
- application pool 40
- amending identity used by 180
- Application Pool property 498–546
- applications. *See also* client applications
- building to host WCF service 58–76
- ApplyDispatcherBehavior method 411, 412
- array length in messages
- WCF runtime limits on 485
- arrays
- data contracts allowing user to send 241
- ASP.NET 50
- ASP.NET client application
- testing 655–656
 - WCF service to support 647–656
- ASPNETDB.MDF database 175
- failure when accessing 180
- ASP.NET Development Server 24, 27, 439, 493
- and discovery 498
 - port 436
 - stopping 481
- ASP.NET Empty Web Site template 580–584
- ASP.NET Membership provider 166–183
- configuring WCF service to use 179–180
- ASP.NET PasswordRecovery control 177
- ASPNETProductsService.cs file 649
- ASP.NET Role Provider 124, 166–183
- configuring WCF service to use 179–180
- ASP.NET Web application
- submitting HTTP DELETE request 576
- ASP.NET Web service
- examining 648–651
 - replacing WCF service with 651–654
- ASP.NET Web Site Administration Tool 166, 174, 192
- ASP.NET Web site, creating to host WCF service 170, 196
- Assign activity 321
- AsymmetricSecurityBindingElement class 77, 404
- AsyncCallback object 448, 507
- asynchronous messaging pattern 333, 337
- asynchronous methods 433
- asynchronous operations 446–457
- in AdventureWorks administrative service 448–452
 - invoking in client application 446–447
 - in WCF service 447–457
 - MSMQ for implementing 463
 - preventing proxy close during 455
- Asynchronous Programming design pattern 507
- AsyncPattern property of OperationContract attribute 447, 448
- AsyncResult class 448

- AsyncResult.cs file 508
 - Atom Publishing Protocol 585, 587
 - authenticated user, identifier for 191
 - authentication 7, 121, 165
 - with ASP.NET Membership Provider and Role Provider 166–183
 - claims-based, in federated environment 643–646
 - for NetTcpBinding binding 151–152
 - service messages with certificate 195–202
 - transport-level 125
 - users with certificates 184–202
 - in Windows environment 123–124
 - of Windows users 146–152
 - authority, verifying 124
 - authorization 122–123, 165
 - identity to determine 625
 - implementing custom 634
 - testing for WCF service 158–163
 - of users 152–160
 - AuthorizationContext property of service security context 633–634
 - Authorization.ServiceAuthorizationManager property 635
 - Authorization Store Role Provider 124
 - autonomy of services 42
- B**
- backward compatibility 238
 - Base64 encoded string 477
 - base address 56
 - Basic256 encryption algorithm 128
 - Basic authentication 150
 - configuring BasicHttpBinding binding to implement 147–148
 - BasicHttpBinding binding 78, 79, 83, 135, 360, 478, 530, 543, 656
 - and client callbacks 606
 - configuring
 - for message-level security 142–143
 - for WCF service 147–148
 - for Windows authentication use 150–151
 - default 33
 - properties 82
 - and sessions 278
 - SOAP messaging format for 429
 - TransferMode property of 490
 - transport-level security
 - for WCF client 136–137
 - for WCF service 136–137
 - BasicHttpContextBinding binding 79, 536, 540, 543
 - for ChannelFactory object 534
 - and sessions 278
 - BeforeSendReply method 409
 - BeginCalculateTotalValueOfStock method 448
 - BeginOpen method 57
 - behavior extension element
 - defining 414–419
 - BehaviorExtensionElement class 414
 - behaviors
 - creating custom 411–414
 - creating for ShoppingCartService service 412–414
 - in WCF Service Model 401–403
 - BehaviorType property 414
 - bidirectional communications 606
 - “big-endian” 32-bit processor 4
 - binary data, converting for transmission 477
 - binary encoding 46, 466
 - binary encoding channel 401
 - BinaryMessageEncodingBinaryElement class 77
 - BindingConfiguration property 136, 439
 - binding elements 33, 77
 - bindings 46, 400
 - client callbacks and 606
 - for COM+ component 664
 - composing channels into 403–408
 - configuring 81–82
 - IIS to support SSL 167–169
 - for WCF service 171–174
 - custom 403
 - for ShoppingCartClient application 395–397
 - for ShoppingCartService service 392
 - programmatically creating and using 405–409

blocking problems, one-way request for resolving

- default maximum received message size 490
- and endpoints 76–84
- order of elements 403–404
- predefined 77–80
- properties for message queues 459
- SendTimeout property of client 435
- and sessions 278
- and transaction support 360
- and WS-ReliableMessaging protocol 389
- blocking problems, one-way request for resolving 443–447
- Body property of TransactedReceiveScope activity 373
- boundaries 42
- Breakpoint command for workflow activity 312
- broadcasting shutdown message 500
- broker model for publishing and subscribing 620
- Browse And Select A .Net Type dialog box 306
- Buffered value for TransferMode property 382, 487
- BufferRemaining element 387
- buffers
 - memory for 487
 - for messages 387
- Build Deployment Package wizard 325
- business analyst 295
- business logic of operations, amending 212–215
- business processes 295
- business-to-business solutions 376

C

- caching announcement requests in dictionary collection 517
- Caching Service, AppFabric 55
- CalculateTotalValueOfStock operation
 - in AdventureWorksAdmin service 447–457
 - invoking asynchronously 454–457
 - invoking in client application 452–454
- CallbackBehavior attribute 606

- CallbackContract property of ServiceContract attribute 600
- callback contracts
 - adding to ProductsService service 607
 - defining 600–601
 - eventing mechanism implemented with 614–622
 - implementing operation in 601–604
 - invoking operation in 604–605
 - for publication and subscription service 620
 - to notify client of one-way operation outcome 606–614
- callback method 434, 448
- callbacks 333. *See also* client callbacks
- CancellationHandler property 376
- CanCreateInstance property 279, 286, 340, 346
 - of Receive activity 333
- catch handlers, in client application 108
- certificates
 - authenticating service messages with 195–202
 - client application required to authenticate using 186–187
 - client applications modified to authenticate with 187–188
 - code for WCF client to override validation checking 141–142
 - creating to identify users 188
 - exporting 188–189
 - importing to Client Certificate Store 198–199
 - importing to Trusted People store 188
 - investigating identifiers of users authenticated with 190–191
 - scripts for managing 189
 - thumbprint of 138
 - updating client applications to send 190
 - for user authentication 184–202
 - WCF runtime determination of validity 141
- Certificates dialog box 193
- CertificateValidationMode property 186
- certification authority (CA) 138, 184, 185
- certmgr console
 - vs. certmgr utility 189

- certmgr utility 188, 193, 200, 635
- ChainTrust 186
- ChangePrice operation 600
- ChangeStockLevel custom activity 375
- ChangeStockLevel method 183
 - testing 36
- ChannelDispatcher object 401, 408, 524–526
 - ServiceThrottle property of 466, 467, 469
 - and timing of servicing session 473
- channel factory, for ShoppingCartServiceRouter service 534
- ChannelFactory object 401, 419
 - BasicHttpContextBinding object for 534
 - for connecting to ProductsService service 420–424
 - creating client-side 428
- ChannelListener object 400, 408
- channels 47
 - basing type on service contract 422
 - composing into bindings 403–408
 - for transactions and reliable sessions 394
 - in WCF Service Model 400–401
- channel shapes 428
- channel stack 47, 77, 204, 392, 400, 524–546
 - constructing 428
 - instantiating 419
 - interaction with service instance 48
- channel variable, invoking methods through 422
- checkout_Click method 292
- Checkout method of ShoppingCartServiceImpl class 250
- Choose Data Source dialog box 11
- claims-based security 626
 - configuring ShoppingCartService service for 629–632
 - implementing 626–641
- claimset 627
- Claim Type Element Editor dialog box 630
- class
 - Entity Framework generation of 12
 - for implementing service contract 45
- Class Library template 10
- client applications
 - ASP.NET, testing 655–656
 - authentication
 - with certificates 186–187
 - authentication of service 122
 - binding configuration for 33
 - building 30–38
 - callback contract for notification of one-way operation outcome 606–614
 - certificate for authenticating of service to 185
 - communication with service 5, 49
 - configuring
 - to authenticate WCF service 199–200
 - to connect with TCP 52
 - to use identity provider 642–643
 - transaction flow to ShoppingCartService service 364
 - connecting to service programmatically 419–426
 - correlating with service instances 346–347
 - data formatting by 203
 - disconnecting 425
 - endpoints for 81
 - fault information use 99
 - for ASP.NET Web services
 - examining 648–651
 - invoking asynchronous operation 446–447
 - invoking CalculateTotalValueOfStock operation 452–454
 - modifying
 - to authenticate with WCF service by using certificate 187–188
 - to capture announcement messages 501
 - to catch SOAP faults 96
 - to connect to updated WCF service 181–182
 - to discover ProductsService service 495–498
 - to send probe requests to discovery proxy 519–521
 - platforms for 647
 - for ProductsSales REST Web service 565–570
 - proxy class regenerated for 103–107
 - reconfiguring 41

- reconfiguring to connect to HTTP endpoint 74–75
- regenerating proxy class and updating 229–231
- reliable messaging in 382
- reliable sessions enabled 381
- request processing 47–49
- running 38
- for SalesData service testing 588–592
- sending large binary data objects to 478–484
- sending message and processing response in 427–431
- sending messages to message queue from 460–461
- service availability by host application 46
- ShoppingCartService service configured to flow transactions from 360
- streaming data 487–488
- testing 424–431
- for testing AdventureWorks Administrative Operations service 440
- for testing ShoppingCartService service 253–257
- trace messages from 383–390
- transaction creation in 361–364
- trustworthiness 165
- unexpected messages from 116–118
- updating
 - after breaking change to service contract 223
 - to send certificate to WCF service 190
 - to subscribe to event 618
- verifying authentication of WCF service 200–201
- for WCF Data service
 - exception handlers 597–598
- WCF data service consumption 587–595
- WCF service configuration to authenticate with to Localhost certificate 196–198
- for workflow service
 - creating 313–316
 - testing 317
- ClientBase abstract class 425–426
- client binding, SendTimeout property of 435
- client callbacks 599, 600–606
 - bindings and duplex channels 606
 - reenetrancy and threading in 605–606
- Client Certificate Store, importing certificate to 198–199
- client communications, controlling 419–431
- ClientCredentials property 151
 - of proxy object 149
- ClientCredentials.UserName property of proxy object 182, 190
- clientEndpointName variable 345
- client endpoint, vs. service endpoint 71
- client library, WCF Data Services 587
- client/server model of processing 599
- client-side ChannelFactory object, creating 428
- Closed event of ServiceHost class 57
- Closed state of ServiceHost object 114
- Close method
 - of proxy object 36
 - of ServiceHost object 57
- Closing event of ServiceHost class 57
- closing service host console window 350
- Closing state of ServiceHost object 114
- cloud computing platform 8
- CodeActivity class 302
- code activity, generating for operation 314
- COM+ 2
- COM+ Application Install Wizard 659
- COM+ applications
 - as WCF service 657–668
 - configuring 661–665
 - testing 665–668
- COM+ catalog, deploying Products COM+ application to 657–661
- COM+ Component Install Wizard 660
- COM+ Integration Wizard 661
- CommittableTransaction object 362
- communications
 - bidirectional 606
 - channel stack and 204
 - inter-process 2–9
- compatibility 43

- CompensableActivity activity 376
- compensating transactions 376
- Complete method of TransactionScope object 363, 366
- CompletesInstance property 280, 286
- Component Object Model (COM) 2
- Component Services console, WS-Atomic-Transaction configuration tab 370
- CompositeType class 16
- computer, global configuration settings 83
- Computer Management console 154
- ComSvcConfig tool 657
- ConcurrencyMode property 258, 445
 - of ServiceBehavior attribute 443
- ConcurrentDictionary class 502
- concurrent sessions, restricting maximum number of 257
- concurrent users, and data changes 353
- confidentiality of communications 122
- Configuration-Based Activation 29
- configuration files
 - for completing tasks 399
 - support for configuring behavior 414
- ConfigurationName property, of workflow service 298
- Connection Properties dialog box 11
- connection string
 - for persistence database 349
 - for ProductsEntityModel assembly 310
 - for ShoppingCartHost project 286
- <connectionStrings> element, in Web.config file 27
- consistency, transactions for 351–377
- Contact entity 551
- content-based routing 523
- Content Definition dialog box 299, 321, 340
- context-based correlation 336
- ContractFilter property, of EndpointDispatcher object 526
- contract-first development 16
- contracts
 - defining 16–18
 - sharing by services 42
- Contract Type Browser 73
- cookies 243
- correlation handle 335
- Correlation Initializers property, of Receive activity 335
- corruption of messages
 - preventing 122
- Create A New Binding dialog box 127, 129, 172, 181, 360, 392
- Create A New Service Wizard 458
- Create A New Standard Endpoint dialog box 494
- CreateBehavior method 414
- CreateBufferedCopy method 410
- CreateChannel method 422
- CreateCustomer method 571
- Created state, of ServiceHost object 114
- CreateMessage method of Message class 429
- Create New Role page 178
- Create New SQL Server Database dialog box 283
- CreateSecureConversationBindingElement method 407
- Create Self-Signed Certificate dialog box 167
- CreateSequence message 384
- CreateSequenceResponse message 384
- Create User page 176–177
- credentials of user 172
- C# source file, generating 106
- Current.PrimaryIdentity.Name property, of ServiceSecurityContext object 191
- Current.PrimaryIdentity property 183
- Current property, of OperationContext class 367
- CurrentStockLevel method 22, 22–23
 - testing 35
- custom activity 341
- custom application, hosting workflow service in 328–332
- custom AuthorizationManager 635
- custom behaviors 402
 - creating 411–414
- CustomBinding object 403
- custom bindings, programmatically creating and using 405–409
- custom channels 400
- Custom Demux sample 527

D

database

- custom code activities for querying 301
- verifying updates 368
- database deadlock 357
- DatabaseFault object 102
 - Visual Basic for creating 321
- database management system 2
- Database property
 - of FindProduct activity 308
 - of ProductExists activity 307
- data contract 16
 - adding class 17
 - defining for WCF service 16–17
 - serialization and security 241–242
- DataContract attribute 17, 224
 - Namespace property 228
- data contracts 203
 - adding field and examining default value 237–238
 - adding new members 231
 - adding WCF service operation for investigating serialization 234–236
 - appearance 224
 - changing namespace for 228–229
 - compatibility of versions 238–242
 - data member name change impact on 227
 - modifying 224–242
 - sequence of members 232
 - changing 225–228
 - versioning strategy 231
- data contract serializer 233
- data format for file sharing 4
- data members, name change impact on data contract 227
- data queries, with REST Web service implementation 549–570
- DataServiceConfiguration class 582
 - SetEntitySetAccessRule method 582
- DataServiceContext class 589
 - HTTP message generation 596
 - SaveChanges method 595
- DataServiceException exception 598
- DataServiceQueryException exception 598

- DataServiceQuery properties 589
- DataServiceRequestException exception 598
- DataServices class 584
- data store for durable service 279
- DataSvcUtil utility 588
- data transmission. *See* Message Transmission Optimization Mechanism (MTOM)
- data variable, workflow problems from renaming 298
- db_owner role, in AdventureWorks database 41
- DCOM (distributed COM) 2
- deactivating service instance, selectively controlling 270–271
- deadlocked database 357
- debugging
 - output messages for 438
 - workflow service 312
- decrementStockLevel method 356
- decryption, and performance 465
- DefaultAlgorithmSuite property 77
- default basicHttpBinding binding 33
- default endpoints 82–84
- default service, code for 14
- default value
 - of field added to data contract 237–238
 - for service throttling 472
- Delegation value for AllowedImpersonationLevel property 162
- DeleteCustomer method in ProductsSales class 575–576
- DeleteCustomer operation 572
- DELETE requests (HTTP) 570, 576
- delivery models for publishing and subscribing 620
- delivery of messages, verifiable 122
- DeliveryRequirements attribute 390
- Denial of Service attacks 241, 484, 486, 490
 - one-way operations and 446
- deploying WCF service
 - in IIS without .svc file 29
 - to IIS 39–41

- Description property of ServiceHost object 413
- deserialization 14
- developer 295
- dictionary collection, caching announcement requests 517
- digital signature 122
- Direct Internet Message Encapsulation protocol (DIME) 478, 484
- disconnecting client applications 425
- disconnecting from WCF service 36
- DiscoverableProductsService Web application 498–546
- discovery 491–522
 - ad hoc 492–499
 - configuring services to support 492
 - handling service announcements 499–506
 - multicast message from service supporting 500
- DiscoveryClient class 496
 - Find method 496–497
 - instantiating object 520
- discovery endpoint
 - creating 517
 - for listening for Probe requests 493
- discovery proxy 507–522
 - configuring ProductsService service to send announcement messages to 519
 - implementing 508–517
 - modifying client application to send probe requests to 519–521
 - testing 521–522
- DiscoveryProxy class 507
- DispatchRuntime object, OperationSelector property of 527
- distributed environment, security in 6
- distributed transaction 351–377
- Distributed Transaction Coordinator (DTC) 352
- DivideByZeroException 111
- DNS servers
 - hacker's rerouting of messages 195
- doClientWork method, in ShoppingCart-Client project 468
- Documents folder 11

- DoWork method 604
- duplex channels 428
 - client callbacks and 606
- DurableOperation attribute 279, 285
- Durable property, for message queue 459
- durable service
 - ShoppingCartService service reconfiguration as 284–288
 - state maintenance with 279–294
 - testing 292–294, 349–350
 - workflow 347–350
- DurableService attribute 279, 285, 529
- Duration property
 - of FindCriteria object 497
- Dynamic Ports property 436

E

- Edit WCF Configuration command 68
- EmitDefaultValue property of DataMember attribute 238
- encoding channel 392, 400
 - in channel stack 48
- encoding format of messages 46
- encoding mechanism, and performance 466
- encrypted message, sender authentication 184
- encryption 122, 125
 - message-level or transport-level 170
 - and performance 465
- EndAsyncSleep method 452
- EndCalculateTotalValueOfStock method 448
- EndOpen method 57
- EndpointAddressFilterMessage object for ChannelDispatcher object 527
- EndpointAddressMessageFilter class 526
- EndpointAddress objects
 - code creating 428
 - collection of 496–497
- "An endpoint configuration for contract... could not be loaded" message 53
- EndpointConfiguration property 314
- EndpointDiscoveryMetadata object for discovery proxy 507
- EndpointDiscoveryMetadata property 503

EndpointDispatcher objects

EndpointDispatcher objects 401, 524–526
 and filters 526–546
 for channel stack 525

EndpointNotFoundException exception 90

endpoints
 associating multiple with same URI 401
 and bindings 76–84
 for client application 81
 contents 32
 default 82–84
 discovery, for listening for Probe requests 493
 for client application for REST service 569

HTTP, reconfiguring client application to connect to 74–75
 listening to 333
 metadata, for service 30
 reconfiguring service to support multiple 72–76
 specifying for service listening 26
 system-defined or application-defined 518
 TCP address specified 53
 testing 76
 WCF runtime determination of default values 68

Enter key, code to wait for 71

Enterprise Services 3, 8

Entity Data Model Wizard 11
 application configuration file from 12

Entity Framework 10
 and data on demand 21
 model 547, 551

error checking in methods 555

error messages, from message processing failure 48

eventing mechanism, callback contract to implement 614–622

Event Sinks 622

Event Sources 622

ExactlyOnce property
 for message queue 459

exception handlers
 exceptions ignored in 95
 in client application for WCF Data service 597–598

order of 108

Exception object, InnerException property 95

exceptions
 client application failure with 41
 detecting and handling 93
 in service host applications 114–118
 lack of details for security 111
 reporting unanticipated 110–113
 thrown by WCF runtime 272
 WCF configuration to send details 112

Execute method of activity
 return value from 303

exporting
 certificates 188–189
 WCF service certificate 198–199

eXtensible Markup Language. *See* XML (eXtensible Markup Language)

Extension Configuration Element Editor dialog box 416, 545

<extensions> element in ShoppingCartHost App.config file 418

F

“Failed to generate a user instance of SQL Server due to a failure...” message 180

FaultContract attribute 318
 adding or removing 223
 in service contract 100–101

Faulted event, of ServiceHost class 57

Faulted state
 for ServiceHost object 115
 method to attempt to determine cause 115

FaultException class 94

FaultException<> exception 318

FaultException handler
 testing 97–99

FaultException object 110
 based on SystemFault object 103
 catch handler for 96
 limitations 99

fault handling
 in host application 115–116
 in ProductsWorkflowService service

- adding 318–323
 - testing 324
 - in workflow service 317–325
- federated environment
 - claims-based authentication in 643–646
- Fielding, Roy 547
- file format for file sharing 4
- file name, changing for file with service
 - contract 17
- filters
 - defining rules for routing messages 544
 - EndpointDispatcher object and 526–546
- filter table, for RoutingService class 539
- FindCriteria class, IsMatch method 504
- FindItem activity 338, 341
- Find method, of DiscoveryClient
 - class 496–500
- FindPrivateKey utility 197
- FindProduct activity 307–308
 - Database property 308
 - implementing 301–305
 - ProductNumber property 308
- FindProduct class 304
- FindResponse object 496
- FindService method, adding to ProductsServiceProxy class 510
- forwarding messages 523
- “Four Tenets of Service Orientation” 42
- front-end service, routing message by 523

G

- garbage collector 257, 425
- GenerateDailySalesReport operation 458
- GetAllCustomers method in ProductsSales
 - class 557, 566
- GetAllOrders method 552
- GetBody<> method 410
- GetCallbackChannel method 604
- GetCustomerForOrder method in ProductsSales
 - class 553, 556, 566
- GetCustomer method in ProductsSales
 - class 558, 566
- GetData method 15, 16
- GetDataUsingDataContract method 15, 16
- GetItemFromDatabase activity 339

- GetOrder method 553, 566
- GetOrdersForCustomer method in ProductsSales
 - class 558, 566
- GetPhoto operation, in ShoppingCartPhotoService
 - service 478
- GetProduct operation 21, 22, 299–301, 315, 323
 - implementing logic 305–309
 - workflow defining 309
- GetShoppingCart method 250, 267, 359
- global configuration settings for computer 83
- global environment, security and privacy
 - in 6–7
- groups
 - adding user account 424
 - creating 154–155

H

- hackers 122, 165, 195
 - and replay attacks 390
- handle variable, workflow use of 298
- handshaking mechanism, for security context token 407
- hard-coding user certificate details 194
- hash 184
- help page for WCF service 28
- hijacking shopping cart 262
- host applications
 - building 63
 - configuring for WCF Service to support TCP 51–52
 - fault handling 115–116
 - for ProductsSales REST Web service 559–561
 - for ProductsServiceHost application 70
 - for ProductsServiceProxy service 517–518
 - service availability to client applications 46
 - service configuration information for 24
 - for ShoppingCartService service 251–253
 - for ShoppingCartService workflow service 343–345
 - tasks required by 45

- unexpected message handling 116–118
 - for Web services 49
 - for workflow service
 - in custom application 328–332
 - in IIS 325–328
 - HostController class
 - adding variable 65
 - handleException method 65–66
 - start_Click method 66
 - stop_Click method 67
 - HTTP
 - default binding for transport 83
 - for listening to port 75
 - reserving port 75–76
 - transport channel 401
 - verifying non-use 54
 - HTTP DELETE requests 570
 - submitting 576
 - HTTP endpoint 205
 - adding to WCF service 72–74
 - configuring 135
 - configuring with SSL certificate 137–141
 - reconfiguring client application to connect to 74–75
 - HttpGetEnabled property
 - of serviceMetadata behavior 85, 118
 - HTTP GET requests, REST operation response to 552
 - HttpGetUrl property, of serviceMetadata element 56
 - HTTP MERGE message 596
 - HTTP POST requests 570
 - HTTP PUT requests 570, 596
 - HTTP service protection 142–145
 - at transport level 135–142
 - HttpsGetEnabled property 172
 - HttpsGetUrl property
 - of serviceMetadata element 56
 - HTTPS protocol 125, 195
 - and transport-level security 137, 172
 - HttpStatusCode enumeration 555
 - HttpsTransportBindingElement class 77
 - HTTPS transport channel 401
 - HTTPS transport, configuring 370
- I**
- IAdventureWorksAdmin interface 437
 - IAsyncResult design pattern 57, 446, 447
 - IAsyncResult object 507
 - IChannelFactory object 534
 - IContextManager object 290, 291
 - IContractBehavior interface 411
 - identification of users 625
 - Identification value for AllowedImpersonationLevel property 162
 - identifying service 201–202
 - <identity> element 201
 - identity, proof of 121. *See also* authentication
 - identity provider
 - client application configuration to use 642–643
 - for Windows CardSpace 641–643
 - IDispatchMessageInspector interface 409
 - IDispatchOperationSelector interface 527
 - IDisposable interface 604
 - IDuplexSessionRouter interface 541
 - IEndpointBehavior interface 411
 - IExtensibleDataObject interface 240
 - IInstanceProvider interface 474
 - IIS DefaultAppPool 40
 - ImpersonateCallerForAllOperations attribute 161
 - impersonation
 - of services, preventing 122
 - to access resources 160–162
 - Impersonation value for AllowedImpersonationLevel property 162
 - Import Application Package wizard 326
 - importing
 - certificates to Trusted People store 188
 - certificate to Client Certificate Store 198–199
 - WCF service code into IIS Web site 170
 - InactivityTimeout property 381, 383
 - IncludeExceptionDetailInFaults property 112, 439
 - information cards for Windows CardSpace 627, 628

- creating 637
 - InitData property, for trace information 132
 - InitializeService method 580
 - InnerChannel property 290
 - InnerException property of exception 95, 320
 - input argument 303
 - input channel 428
 - inspecting messages 408–419
 - installing
 - Windows Process Activation Service (WAS) 50
 - Windows service 88–89
 - “instance context” for services 602
 - InstanceContextMode property 48
 - of ServiceBehavior attribute 258–261, 267
 - InstanceContext object 48, 603
 - InstanceData table 293
 - instance pooling 474
 - integrity, message-level 126
 - Internet 3
 - as hostile environment 166
 - Internet Explorer
 - data display from AdventureWorks database 585–587
 - HTTP GET requests submitted from 561
 - testing WCF service with 27–30
 - turning off feed-reading view 585
 - Internet Information Services (IIS) 49
 - configuring bindings to support SSL 167–169
 - default port for HTTP protocol 73
 - deploying ProductsService service 498–500
 - deploying WCF service 39–41
 - without .svc file 29
 - importing WCF service code into Web site 170–171
 - stopping and restarting 197
 - workflow service hosting 325–328
 - Internet Information Services Manager 40, 51, 506
 - InternetProductsService Web application,
 - creating 169
 - inter-process communications 2–9
 - InvalidOperationException exception 53, 605
 - IOperationBehavior interface 411
 - IProductService.cs file 420, 651
 - IProductsService interface 652, 658
 - IRequestChannel class
 - Request method of 536
 - IRequestReply interface 541
 - IRequestReplyRouter interface 542
 - IServiceBehavior interface 411
 - IService.cs file 15–16
 - IService interface 15–16
 - IShoppingCartPhotoService interface 478
 - IShoppingCartService.cs file
 - TransactionFlow attributes 355
 - IShoppingCartService interface 468
 - ISimpleProductsService.cs file 427
 - ISimpleProductsService interface 427
 - ISimplexDatagramRouter interface 541
 - ISimplexSessionRouter interface 541
 - IsInitiating property of OperationContract attribute 272
 - IsInRole method 157
 - of System.Web.Security.Roles class 183
 - IsMatch method of FindCriteria class 504
 - IsolationLevel property 376
 - isolation levels for transactions 354, 362
 - IsRequired property of DataMember attribute 238
 - IssuedTokenType property 630
 - IsSystemendpoint property 518
 - IsTerminating property of OperationContract attribute 272
- ## J
- JavaScript 5
 - JavaScript Object Notation (JSON) 6, 548, 552
 - Java Web services 647

K

“keep alive” message 383
 Kerberos protocol 123
 Kind property 494

L

large binary data objects, sending to client application 478–484
 LastMessage message 388
 listener object 132
 Listener Settings dialog box 207, 593
 ListProducts method 20, 118, 208
 in ProductsServiceImpl class 95
 testing 34
 load-balancing 465, 523
 algorithm 535
 for router 528
 Load method, of XamlServices class 330
 Localhost certificate
 removing from certificate store 200
 WCF service configuration to authenticate to client applications with 196–198
 LocalMachine certificate store, removing certificate from 200
 locked resources, transactions and 376
 locks, for data-modification transactions 354
 LogEntireMessage property 131, 382
 LogMalformedMessages property 382
 LogMessagesAtServiceLevel property 131
 LogMessagesAtTransportLevel property 131
 logs
 for exceptions and errors from service failure 89
 minimizing overhead 207
 long-running transactions 376–377
 lost messages 380

M

machine.config file 83
 MainWindow class 345
 MainWindow.xaml.cs file 345
 makecert command 654

makecert utility 138, 188, 196, 631
 mapping client certificates to Windows accounts 194
 MatchAllMessageFilter object 532
 Match method 526, 527
 MaxArrayLength property 485, 486
 MaxBufferSize property 486
 MaxBytesPerRead property 485, 486
 MaxClockSkew property 393
 MaxConcurrentCalls property, for serviceThrottling 466, 471, 473
 MaxConcurrentInstances property, for serviceThrottling 466, 471
 MaxConcurrentSessions property, for serviceThrottling 467, 471, 473
 MaxConnections property of binding configuration 257
 MaxDepth property 485, 486
 MaxNameTableCharCount property 485, 486
 MaxReceivedMessageSize property 486, 490
 MaxResponseDelay property 494
 MaxResults property of FindCriteria object 497
 MaxStringContentLength property 485, 486
 membership provider 180
 memory, service requirements 475
 MERGE message (HTTP) 596
 MessageBox class 437
 Message class, CreateMessage method of 429
 MessageClientCredentialType property 152, 181, 186
 MessageEncoding property of binding configuration 478, 482, 483
 MessageFilter abstract class 526
 MessageFilter class 539
 message headers, and routing to other services 528
 message-level encryption 170
 for NetTcpBinding binding
 for WCF service 127
 in WCF client 129–131
 message-level security 126, 421

- BasicHttpBinding binding configured
 - for 142–143
- message queue support for 460
- and streaming 490
- Web site to host WCF service implementing 196
- message patterns
 - for workflow service 332–337
- message queues 433, 457–463
 - sending messages from client application 460–461
 - WCF service use of 458–460
- message queuing URI, format for 459
- messages. *See also* reliable messaging;
routing
 - buffers for 387
 - configuring tracing for WCF service 131–132
 - displaying in WCF service 235
 - display of announcement 506
 - host application handling of unexpected 116–118
 - inspecting 408–419
 - “keep alive” 383
 - preventing corruption 122
 - routing based on contents 539
 - sending programmatically 427–431
 - sending to client display 214
 - sequence order 379
 - transactions and 372
 - verifying sender identity 195
 - volatile 457
- MessageSecurityException exception 148
- Message security mode for binding 142
- message sequence number 385
- Message Transmission Optimization Mechanism (MTOM) 400, 466, 476–487
 - configuring ShoppingCartPhotoService service to transmit messages 482–484
 - controlling message size 484–487
- messaging pattern
 - interfaces specifying 428
- metadata
 - adding to services collection 509
 - publishing 56
 - for WCF service description 29
 - Web service publishing of 26
- metadata endpoint
 - for service 30
- methods, overloading 219
- Microsoft
 - Component Object Model (COM) 2
 - Microsoft Authorization Manager 124
 - Microsoft Message Queue (MSMQ) 8, 49, 457
 - asynchronous operations implementation 463
 - Microsoft .NET Framework 3.51. *See* .NET Framework
 - Microsoft Silverlight 5
 - Microsoft Transaction Server 2
 - Microsoft Windows Network Monitor 484
 - minFreeMemoryPercentageToActivateService attribute 475
 - MsmqIntegrationBinding binding 80
 - and reliable messaging 389
 - and sessions 278
 - MTOM. *See* Message Transmission Optimization Mechanism (MTOM)
 - multicast message, from service supporting discovery 500
 - MultipleFilterMatchesException exception 527
 - multiple replies to the client 337
 - multipleSiteBindingsEnabled property 173, 439
 - Multipurpose Internet Mail Extension (MIME) specification 477
 - multi-threading
 - vs. reliable sessions 445
 - transactions and 372
- N**
- named pipes 2, 49, 85, 125
 - restricting maximum number of concurrent sessions 257
- Name property
 - of OperationContract attribute 219

names

- for bindings 128
- for interface 611
- for operations, changing 222
- namespace parameter 106
- Namespace property of DataContract attribute 228
- namespaces
 - changing for data contract 228–229
 - statements referencing 14
 - for WCF service class 25, 27
- NegotiateServiceCredential property 186
- nested data, data contracts allowing user to send 241
- nested TransactionScope object 363
- .NET Framework 3, 37, 50
 - assembly 47
 - common language runtime (CLR)
 - exceptions 94–113
 - XML serialization features 224
 - .NET Framework garbage collector 257, 425
 - .NET Framework Global Assembly Cache 659
- NetMsmqBinding binding 80, 360, 459, 463
 - and reliable messaging 389
 - and sessions 278
- NetNamedPipeBinding binding 80, 369, 382
 - and client callbacks 606
 - creating 87
 - and sessions 278
- NetPeerTcpBinding binding 80, 360, 599
 - and sessions 278
- netsh utility 75, 205, 256, 655
 - for SSL configuration 138
- NetTcpBinding binding 78, 80, 369, 381, 382
 - authentication mechanism for 151–152
 - and client callbacks 606
 - configuration 82
 - configuring to require Username authentication 150
 - message-level encryption 127

- for WCF service 127–129
 - in WCF client 129–131
- and reliable messaging 389
- and sessions 278
- NetTcpContextBinding binding 80, 344, 346, 369, 536
 - and sessions 278
- network analyzers 122, 484
- network operating systems (NOS) 2
- network resource
 - account access to 75
- NETWORKSERVICE account 180, 197
- networks, reliability issues 47, 379
- New Client Element Wizard 254
- New Group dialog box 154
- New Private Queue dialog box 462
- New Project dialog box 58
- New Service Element Wizard 68, 251, 330, 438
- New User dialog box 154
- New Web Site dialog box 170
- nonces 391
- Notepad, to view XML file 268

O

- OASIS (Organization for the Advancement of Structured Information Standards) 7, 380
- ObjectContext class 580
- OfflineAnnouncementReceived event 503
- OLE transactions, implementing 352–369
- OnBeginFind method 508
 - overriding 515
- OnBeginOfflineAnnouncement method 508
 - overriding 514–515
- OnBeginOnlineAnnouncement method 507
 - overriding 513–514
- OnBeginResolve method 508
 - overriding 516
- OnEndFind method 508, 515
- OnEndOfflineAnnouncement method 508–509
 - overriding 514–515

- OnEndOnlineAnnouncement method 507
 - overriding 513
 - OnEndResolve method 508
 - overriding 516
 - one-way messaging 336
 - one-way operations 433, 434–446
 - callback contract for client notification of outcome 606–614
 - and Denial of Service attacks 446
 - effects 434–435
 - implementing 436–445
 - recommendations 445–446
 - and timeouts 435–436
 - and transactions 435
 - one-way request, for resolving blocking problem 443–447
 - OnlineAnnouncementReceived event 500
 - subscribing to 503
 - OnPriceChanged operation 600
 - Opened event, of ServiceHost class 57
 - Opened state, of ServiceHost object 114
 - Opening event, of ServiceHost class 57
 - Opening state, of ServiceHost object 114
 - Open method
 - to listen for requests 56
 - of WorkflowServiceHost class 330
 - OperationBehavior attribute 270
 - Impersonation property 160
 - TransactionAutoComplete property of 371, 372
 - TransactionScopeRequired attribute of 362
 - OperationContext class
 - Current property 367
 - OperationContext.Current property 604
 - OperationContract attribute 18, 47, 272, 552
 - Action property of 532
 - AsyncPattern property 447, 448
 - Name property 219
 - properties 428
 - ProtectionLevel property 206, 208
 - operations
 - adding to service contracts 222
 - adding to WCF service 218–222
 - asynchronous 446–457
 - in AdventureWorks administrative service 448–452
 - invoking in client application 446–447
 - in WCF service 447–457
 - preventing proxy close during 455
 - removing from service contract 222
 - OperationSelector property of DispatchRuntime object 527
 - operations in ShoppingCartService service
 - controlling sequence 273–274
 - testing 274–294
 - Ordered property 390
 - Organization for the Advancement of Structured Information Standards (OASIS) 7, 380
 - output channel 428
 - Output.config file 106
 - output messages, for testing or debugging 438
 - overloaded constructor, for ProductsServiceClient class 34
 - overloading methods 219
- P**
- parameters
 - adding to existing operation in service contract 216–218
 - changing in operation 223
 - for service contract methods 224
 - passwords 177, 182, 625
 - query of Active Directory to verify 123
 - risks in using 184
 - Windows Integrated Security and 151
 - Peer Channel 599
 - PeerOrChainTrust 187
 - peer-to-peer networking 599
 - PeerTrust 186
 - PerCall instance context mode 259, 260, 474
 - maintaining state with 262–270
 - PerCall service instance context mode 389, 445
 - performance 465
 - of network, multiple endpoints for 72

- PermissiveCertificatePolicy.Enact method
- PermissiveCertificatePolicy.Enact method 181
- PerSession instance context mode 258, 371, 379, 389, 468, 474, 529
 - as default 262
 - ReleaseInstanceMode property and 271
- PersistBeforeSend property 347
- Persistence Provider Arguments Editor dialog box 287
- persistence store
 - creating for SQL Persistence Provider 283–284
 - session state persisting to 290
- personal computers, early applications 1–2
- Pick activity 340
- “Plain Old XML” (POX) 548
- platforms for client applications 647
- pool of service instance objects 474
- ports
 - for ASP.NET Development Web Server 436
 - binding certificate with thumbprint to 140
 - HTTP protocol for listening to 75
 - removing HTTP reservation for 140
- POST requests (HTTP) 570
- “The primary signature must be encrypted” exception 208
- PrincipalPermission attribute 156, 182
- privacy
 - in global environment 6–7
 - message-level 126
 - XML files and 269
- private key 184
 - protecting 201
- private message queue 459
- ProbeMatch message 492
- Probe message 491
 - address for sending 519
 - services responding to 496–497
- Probe requests
 - discovery endpoint for listening for 493
 - modifying client application to send to discovery proxy 519–521
- ProcessMessage method 531, 535
 - signature of 532
- Product class, modifying definition 651
- ProductExists activity
 - Database property of 307
 - implementing 301–305
 - ProductNumber property 307
- ProductNumber property
 - of FindProduct activity 308
 - of ProductExists activity 307
- Products.cs file 106
- ProductsEntityModel assembly 301
- Products property 21
- ProductsSales class
 - DeleteCustomer method 575–576
 - GetAllCustomers method 557
 - GetCustomerForOrder method 556
 - GetCustomer method 558
 - GetOrdersForCustomer method 558
- ProductsSalesModel.edmx file 551
- ProductsSalesProxy class
 - creating instance 568
- ProductsSales REST Web service
 - client application for 565–570
 - extending to support data updates 571–576
 - hosting 559–561
 - implementing 554–559
 - testing 561–565, 577–579
- ProductsSales REST Web service contract 550
- ProductsServiceClient class 34
- ProductsServiceClient proxy, creating instance 497–498
- ProductsServiceHost application 70
- ProductsServiceImpl class
 - ListProducts method 95
- ProductsServiceProxy class
 - adding FindService method 510
 - adding ResolveService method 511
 - adding WaitForAsyncResult utility method 512
- ProductsServiceProxy service, host for 517–518
- ProductsService service
 - callback contract added to 607

- ChannelFactory object for connecting to 420–424
- configuring
 - for ad hoc discovery 492–495
 - to send announcement messages to discovery proxy 519
 - to send announcements 500–501
- deploying to IIS 498–500
- modifying client application to discover 495–498
- rebuilding as WCF Service Library 58–59
- subscribe and unsubscribe operations in 615–618
- testing 505–506, 619
- ProductsWorkflowClient project, adding service reference 314
- ProductsWorkflowService namespace
 - FindProduct class 304
 - ProductData class 302
- ProductsWorkflowService service
 - adding item to 301
 - contents 327
 - creating 296
 - custom application for hosting 329–331
 - fault handling
 - adding 318–323
 - testing 324
 - hosting by IIS, testing 328
 - ProductsService.xamlx file 323
 - testing 310–312
 - testing host 331–332
 - WCF service in 309
- Products.xsd file 104
- Program.cs file 71
 - using statement 33
- protection level of operation, changing 223
- ProtectionLevel property
 - of Receive activity 333
- ProtectionLevel property of OperationContract attribute 206, 208
- proxy class
 - for client application
 - for callback contract 610–612
 - for client application
 - to test ShoppingCartService service 253
 - creating for service 419
 - for communicating with WCF service 49
 - generating 30
 - regenerating 31
 - for client application 103–107, 233
 - and updating client application 229–231
 - viewing code 31
 - proxy object
 - ClientCredentials property 149
 - ClientCredentials.UserName property 182, 190
 - Close method 36
 - code to wait for Enter key before creating 71
 - configuring 36
 - creating 53, 290
 - creating instance 34
 - generating 220
 - opening 77
 - preventing close during asynchronous operation 455
 - regeneration after service contract changes 208
 - public key cryptography 125
 - public key infrastructure (PKI) 184
 - public message queues 459
 - publishing, delivery models for 620
 - Publish Web Site dialog box 39, 498–546
 - pull delivery model 620
 - push delivery model 620
 - PUT requests (HTTP) 570, 596

R

- ReadCommitted isolation level 354
- ReaderQuota properties of bindings 241–242, 485
- ReadUncommitted isolation level 354
- Receive activity 333, 335
 - Correlation Initializers property 335
- ReceiveAndSendReply activity 334, 336
- ReceiveReply activity 334
- ReceiveRequest activity 297–298, 321
- reentrancy in callback operation 605–606

reference, adding to WCF service

reference, adding to WCF service 19

regasm command 370

registering callback 259

relativeAddress element 29

ReleaseInstanceMode property 270

ReleaseServiceInstanceOnTransactionComplete property 371, 372

reliability of communications 47

reliable message protocol 122

reliable messaging 380–390

implementing 389

in client applications, timeout from 382

vs. reliable sessions 379, 389

and streaming 490

ReliableSessionBindingElement class 77

reliableSession binding extension element 394

reliable sessions 444

implementing with WCF 381–390

vs. multiple-threading 445

and replay attacks 390

reliable sessions binding element 406

RemoveItemFromCart method 266, 359

as durable operation 286

RemoveItemFromCart method of ShoppingCartServiceImpl class 249

RepeatableRead isolation level 354

replay attacks 122, 379, 390–397

ReplayCacheSize property 393

ReplayWindow property 393

ReplyAction property for OperationContract attribute 428

reply channel 428

reply messages, correlating with request messages in workflow instance 335–336

Representational State Transfer model (REST) 14

request channel 428

request messages 46

activity to send 334

correlating with reply messages in workflow instance 335–336

Request method of IRequestChannel class 536

Request property

of TransactedReceiveScope activity 373

request/response messaging pattern 433

ResolveService method

adding to ProductsServiceProxy class 511

resources

availability 475

impersonation to access 160–162

locked by transactions 376

and performance 465

service throttling to control use 466–475

response messages 46

response streaming 488

response time of WCF service 467

restoreShoppingCart method 269

REST (Representational State Transfer) model 14, 547–549

REST Web service

for data updates 570–579

querying data by implementing 549–570

service contract for 550

Result property, of ProductExists activity 307

return type of operation

changing 218, 223

return values

from Execute method of activity 303

for service contract methods 224

RevocationMode property 187

Rich Internet Applications (RIAs) 5–6

role provider 180

RoleProviderName property 179

role providers 124

roles 124

for WCF service

defining 174–179

specifying 156–158

round-tripping 238

by WCF runtime 239–242

RouteOnHeadersOnly property 542

Router class

private fields 533

ProcessMessage method 535

ServiceBehavior attribute 533

static constructor for 534

routing 523–539

filters defining rules for 544

routing behavior, SoapProcessingEnabled property of 542
 RoutingService class 523, 539–546
 RoutingService service
 hosting and configuring 540–545
 testing 545–546
 RunClients.cmd command 619
 runtime environment 45

S

SalesData service 587
 testing 593
 client application for 588–592
 SalesDataService class 581–582
 SalesData WCF Data Service, testing 584–587
 SalesOrderHeader entity 551
 sample database, AdventureWorks
 OLTP 9–10
 SaveChanges method of DataServiceContext object 595
 saveShoppingCart method 263, 269
 scalability
 controlling 467
 of workflow services 347
 Schemas.microsoft.com.2003.10.Serialization.Arrays.xsd file 105
 Schemas.microsoft.com.2003.10.Serialization.xsd file 105
 schemas, sharing by services 42
 scope for behaviors 402
 scripts
 for certificate management 189
 to query AdventureWorks database 368
 SecureConversationBindingElement element 406
 Secure Sockets Layer (SSL) 125, 167
 configuring 136
 configuring IIS bindings to support 167–169
 HTTP endpoint configuration with certificate 137–141
 security. *See also* Windows CardSpace
 basics 121–126
 configuring 424–431

data contract serialization and 241–242
 federated 645
 for HTTP service 142–145
 in global environment 6–7
 implementing in Windows domain 127–162
 lack of exception detail and 111
 message-level 126
 and performance 465
 replay attacks 379
 requirements of service 46
 specifying requirements for WCF service operations 205–207
 SQL Server database for information 175
 transport-level 125
 WCF Data Services template and 582
 SecurityAction.Demand parameter 156
 SecurityAction.Deny parameter 156
 Security Assertion Markup Language (SAML) 628
 SecurityBindingElement class 404, 406
 security context token
 handshaking mechanism for 407
 security risk
 hijacking shopping cart 262
 on HTTP connection 75
 security token in Windows CardSpace 628
 SecurityTokenService class 646
 Security Token Service (STS) 642–643
 SelectOperation method 527
 self-issued cards 641
 for Windows CardSpace 629
 self-signed certificate 167
 Send activity 334
 SendAndReceiveReply activity 334, 336
 Send GetShoppingCart Response activity
 PersistBeforeSend property 347
 sending messages programmatically 427–431
 Send RemoveItemFromCart Response activity
 PersistBeforeSend property 347
 SendReply activity 321, 334, 335, 336
 SendReplyToReceiveRequest activity 321, 322

SendResponse activity

SendResponse activity 297–298, 308
 Send SystemFault activity 322
 SendTimeout property of client binding 435
 <SequenceAcknowledgement> block in messages 387
 Sequence activity 321
 for TransactedReceiveScope activity 374
 sequence of messages
 need for order 379
 numbering for 384
 “sequence terminated” SOAP fault message 381
 sequencing operations 243, 271–294
 Sequential Service activity 305, 306
 serializable class 224
 for SQL Persistence Provider 285
 Serializable isolation level 354
 serialization 14
 adding WCF service operation for investigating serialization 234–236
 of data contract, and security 241
 of publicly accessible classes 263
 Server Certificates 167
 service
 certification for authenticating to client application 185
 client application communication with 49
 identifying 201–202
 implementing successful 8
 memory requirements for 475
 preventing impersonation 122
 response to client requests 47
 versioning 211–222
 in WCF Service Model 400–401
 Service Activation Editor dialog box 171
 ServiceActivationException exception 475
 service address 46
 service announcements
 handling 499–506
 serviceAuthorization behavior 153
 ServiceAuthorizationManager class 634
 serviceAuthorization service behavior element 161

ServiceBehavior attribute

 of Router class 533
 ServiceBehavior property 48, 113, 353–377
 ConcurrencyMode property of 443
 InstanceContextMode property of 258–261, 267
 investigating 260–262
 and ReleaseInstanceMode property 270–271
 ReleaseServiceInstanceOnTransactionComplete property 372
 of ShoppingCartServiceImpl class 468
 TransactionAutoCompleteOnSessionClose property 372
 <serviceBehaviors> section in ShoppingCartHost App.config file 418
 Service Configuration Editor 68
 service contract
 for generalized WCF service to accept messages 528
 for router service 531
 sending Probe request 496
 ServiceContract attribute 18, 47
 CallbackContract property 600
 service contracts 16, 45, 203
 adding parameters to existing operation 216–218
 breaking and nonbreaking changes to 222–223
 changes, and proxy regeneration 208
 channel type based on 422
 defining synchronous and asynchronous versions of operation 456
 FaultContract attribute in 100–101
 information on potential faults 99
 in ShoppingCartService namespace 245–246
 ISimpleProductsService interface defining 427
 modifying 204–223
 multiple 220
 for REST Web service 550
 selective protection of operations 205–211

- SessionMode property of
 - updating 211
 - for WCF service 18
- serviceDebug behavior element 402
- service definition file 73
- serviceDiscovery behavior element 493, 501
- service endpoints 46–47
 - vs. client endpoint 71
- service host applications
 - application configuration file 110
 - console window, closing 350
 - exception management 114–118
- ServiceHost class 55–57
 - AddDefaultEndpoints method 82
 - events 57
 - to host WCF service in Windows application 58–76
- ServiceHost object 407
 - Description property of 413
 - opening 57
 - specifying addresses to be listened to 55
 - states and transitions 114–115
- service instance
 - correlating clients with 346–347
 - ID for session 279
 - interaction with channel stack 48
 - selectively controlling deactivation 270–271
- service instance context modes 257–262
- service instance pooling 474
- service messages, authenticating with certificate 195–202
- serviceMetadata element
 - httpGetEnabled property 85, 118
- Service-Oriented Application (SOA), transactions in 351–377
- Service-Oriented Architecture (SOA) 7–9, 42–43, 203
- service-oriented wrappers 295
- service reference, adding to ProductsWorkflowClient project 314
- Service Reference Settings dialog box 446
- services
 - configuring to support discovery 492
 - “instance context” for 602
- services collection
 - adding metadata 509
- service security context, AuthorizationContext property of 633–634
- ServiceSecurityContext class 183
 - Current.PrimaryIdentity.Name property 191
- Service.svc file 25
- ServiceThrottle class
 - properties 466–467
- service throttling 465, 466–475
 - configuring 467–474
 - default values for 472
 - purpose of 474
- serviceThrottling behavior element 471–472
- Service Trace Viewer 396, 481, 483, 594
- Service Trace Viewer utility 145
- SessionMode property of service contract 272
- sessions 445
 - bindings and 278
 - disabling 443
 - for durable service 279
 - reliability
 - implementing with WCF 381–390
 - vs. reliable messaging 379
 - tables for storing information 293
 - transactions and 372
 - in workflow service 337–350
- session state, persisting to persistence store 290
- SetEntitySetAccessRule method of DataServiceConfiguration class 582
- SetEntitySetPageSize method 583
- SetTransactionComplete method 367
- shopping cart 243–244, 337
 - hijacking 262
- ShoppingCartBehavior behavior
 - creating behavior extension element for 415
- ShoppingCartService configuration to use 416–419
- ShoppingCartClient application
 - configuring to use Windows CardSpace to send token 635–637

ShoppingCartGUIClient application

- custom binding for 395–397
- doClientWork method 468
- ShoppingCartGUIClient application 345
 - addItem_Click method 282
 - examining 280
 - poor practices in initial version 283
 - updating 288–292
 - user interface code 281–282
 - version using streaming 489
- ShoppingCartHost application
 - App.config file 529
 - configuring to host ShoppingCartRouterService service 537–538
- ShoppingCartInspector.cs file 409
- ShoppingCartItem class 245, 285, 338
- ShoppingCartPhotoService service 478
 - configuring to transmit MTOM-encoded messages 482–484
 - default process for sending messages 482
 - GetPhoto operation 478
 - version using streaming 489
- ShoppingCartRouter service
 - creating 530–537
 - ShoppingCartHost application configuration to host 537–538
 - testing 538–539
- ShoppingCartService.Activities.cs file 338
- ShoppingCartServiceImpl class
 - AddItemToCart method 265
 - OperationBehavior attribute 354–355
 - code to check stock levels 356
 - GetShoppingCart method 267
 - InstanceContextMode property of the ServiceBehavior attribute 267
 - RemoveItemFromCart method 266
 - as durable operation 286
 - restoreShoppingCart method 264, 269
 - ServiceBehavior attribute of 468
- ShoppingCartService service 529–530
 - applying throttling 468–474
 - change to authorize users based on email address 632–634
 - channels for transactions and reliable sessions 394
 - client application for testing 253–257

- configuring for claims-based security 629–632
- configuring to use ShoppingCartBehavior behavior 416–419
- creating 244
- creating behavior 412–414
- custom binding for 392
- host application 251–253
- message inspector for 409–411
- operations sequence 271–272
 - controlling 273–274
 - testing 274–294
- programmatically creating and using custom binding 405–409
- reconfiguring as durable service 284–288
- reliable sessions enabled 381
- state management 262
 - testing capabilities 267–270
- testing 638–641
 - information cards for 637
- transactions 353
 - configuring client applications for flow 364
 - configuring service to flow from client applications 360
 - modifications for indicating success 366
 - testing 365–369
 - using block 361
 - workflow 338–342
 - configuring as durable service 347
- ShoppingCartService workflow service
 - hosting 343–345
 - testing 345–346
- shutdown message, broadcasting by service 500
- signature 185
 - for MTOM messages 478
- Silverlight. *See* Microsoft Silverlight
- Simple Object Access Protocol (SOAP) 547
- Single instance context mode 259, 261
- Site Bindings dialog box 51, 168
- “small-endian” 32-bit processor 4
- SOAP (Simple Object Access Protocol) 5, 219

- SOAP fault messages
 - one-way operation and 434
 - "sequence terminated" 381
- SOAP faults 94
 - modifying client applications to catch 96
 - strongly-typed 99–110
 - modifying WCF service to throw 101–103
 - WCF client modified to catch 108–110
 - throwing and catching 94–99
- SOAP header, <AckRequested> block 385
- SOAP messages 204, 400
 - for client parameters passed into operation 532
 - in Message object 410
 - MTOM for sending and receiving 476
 - raw format 484
- SoapProcessingEnabled property of routing behavior 542
- SOAP serialization 231
- SOAP specification 94, 429
- sockets 2
- Solution 'ProductsServiceLibrary' Property Pages dialog box 72
- "A specified logon session does not exist..." message 140
- spoofing 122
- spooft service 185
- SQL Persistence Provider 279, 287, 293
 - persistence store for 283–284
 - serializable class for 285
- SqlPersistenceProvider
 - Factory class 287
- SQL Server
 - adding default accounts for application pools 41
 - updating user information in database 192
- SQL Server persistence provider 529
- SslStreamSecurityBindingElement
 - class 404
- standard endpoints 492
 - configuration 494
- starting
 - Internet Information Services (IIS) 197
 - WCF service, code for 65–67
 - Windows service 85–87
- startup projects, multiple projects set as 256
- state information, maintaining between operations 243
- stateless, REST model as 549
- state maintenance
 - with durable service 279–294
 - with PerCall instance context mode 262–270
- testing capabilities of ShoppingCartService service 267–270
 - in WCF service 244–271
 - in workflow service 337–350
- stopping
 - Internet Information Services (IIS) 197
 - WCF service, code for 65–67
 - Windows service 85–87
- StreamedRequest value for TransferMode property 487
- StreamedResponse value for TransferMode property 488
- Streamed value for TransferMode property 488
- streaming data 487–490
 - and client application 487–488
 - message-level security and 490
 - operation design to support 488–490
 - ShoppingCartPhotoService version using 489
- stream upgrades 404
- strongly-typed SOAP faults 99–110
 - WCF client modified to catch 108–110
 - WCF service modified to throw 101–103
- subscribe operations in ProductsService service 615–618
- subscribing, delivery models for 620
- Subscription Manager 622
- svcutil utility 49, 103, 106, 203, 220, 229, 419
 - /async flag 446
- SymmetricSecurityBindingElement
 - class 404
- System.Collections.Concurrent
 - namespace 502
- System.Data.Entity assembly 245

System.Data.Services.EntitySetRights enumeration

System.Data.Services.EntitySetRights enumeration 583

system-defined endpoints
or application-defined 518

SystemFault object
creating 322

FaultException based on 103

System.IdentityModel.Claims.ClaimSet
class 641

System.Net.PeerToPeer namespace 599

System.Runtime.Serialization
namespace 14

System.ServiceModel.Activities
namespace 328

System.ServiceModel assembly, adding
reference to 34

System.ServiceModel.Channels
namespace 405

System.ServiceModel.ClientBase generic
abstract class 425

System.ServiceModel.Discovery.FindCriteria
object 496–546

System.ServiceModel.Discovery
namespace 500, 507

System.ServiceModel.DuplexClientBase
class 601

System.ServiceModel.FaultException object
95

System.ServiceModel namespace 14
predefined bindings in 78

WorkflowServiceHost class 328

System.ServiceModel.Routing
namespace 523

System.ServiceModel.Web assembly 549

System.ServiceModel.Web namespace 14,
559

System.TimeoutException 435

System.Transactions namespace 352

System.Web.Security.Roles class
IsInRole method 183

System.Xml namespace 410

T

TCP endpoint, creating for discovery proxy
connection 520

TCP protocol 49, 72

configuring client application to connect
with 52

configuring host environment for WCF
Service to support 51–52

for discovery 517

protecting at message level 127–135

restricting maximum number of concurrent
sessions 257

TCP transport channel

binding elements to implement 407

templates

Class Library 10

for WCF service 9, 13

for WCF Service Library template 58

Tempuri.org.wsdl file 105

Tempuri.org.xsd file 104

TerminateSequence message 388

testing

AdventureWorks Administrative Operations
service, client applications
for 440

AdventureWorksAdmin queue 462–463

ASP.NET client application 655–656

authorization for WCF service 158–163

ChangeStockLevel method 36

client applications 424–431

for workflow service 317

COM+ applications 665–668

CurrentStockLevel method 35

discovery proxy 521–522

durable service 292–294, 349–350

endpoints 76

FaultException handler 97–99

fault handling

in ProductsWorkflowService service
324

ListProducts method 34

output messages for 438

ProductsSales REST Web service 561–
565, 577–579

ProductsService service 505–506

ProductsService service event 619

ProductsWorkflowService service 310–
312

host for 331–332

IIS host 328

- RoutingService service 545–546
- SalesData service 593
 - client application for 588–592
- SalesData WCF Data Service 584–587
- ShoppingCartRouter service 538–539
- ShoppingCartService service 638–641
 - client application for 253–257
 - information cards for 637
 - operation sequencing 274–294
 - state management capabilities 267–270
 - transactions 365–369
- ShoppingCartService workflow service 345–346
- WCF service 41, 182–185
 - with Internet Explorer 27–30
- Windows host environment 70
- Windows service 89–90
- TestProductsService method 607, 611, 612
- text encoding channel 401
 - binding elements to implement 407
- Thawte 138
- “There is a problem with the website’s security certificate” error 174
- “This operation would deadlock because...” message 605
- threads
 - in callback operation 605–606
 - and service instance lifetime 259
- throughput 467, 475
- thumbprint of certificate 138
- timeouts
 - one-way operations and 435–436
 - from reliable messaging in client applications 382
- time-stamped identifier
 - for replay detection 391
- time-stamp in message 122
- token in Windows CardSpace 628
 - specifying type expected 630
- trace messages from client applications 383–390
- trace output
 - examining 133–135
 - running WCF client and service to examine 145–146
- TraceOutputOptions property
 - for trace information 132
- tracing
 - configuring 382, 593–595
 - information for WCF 131
 - WriteLine statement for 468
- TransactedReceiveScope activity 374
 - properties 373
- transactional requirements of service 46
- TransactionAutoCompleteOnSessionClose property 371
 - of ServiceBehavior property 372
- TransactionAutoComplete property 359
 - of OperationBehavior attribute 355, 371, 372
- transaction channels 47
- transaction flow binding element 406
- transactionFlow binding extension element 394
- TransactionFlow property 360
- TransactionIsolationLevel property 353–377
- TransactionProtocol property 360, 369
- transactions
 - aborted client requests impact 474
 - aborting 357
 - and multi-threading 372
 - COM+ component support for 664
 - configuring client applications for flow to ShoppingCartService service 364
 - creating in client application 361–364
 - enabling in ShoppingCartService service 353
 - for internal consistency 351–377
 - in ShoppingCartService service, testing 365
 - in workflow service 373–377
 - isolation levels for 354
 - long-running 376–377
 - and messaging 372
 - one-way operations and 435
 - ShoppingCartService service configured to flow from client applications 360
 - in WCF service 352–371
- TransactionScope activity
 - workflow client using 375
- TransactionScope object
 - Complete method 363, 366
 - nested 363

TransactionScopeOption parameter

TransactionScopeOption parameter 362
 TransactionScopeRequired property 359–360
 of OperationBehavior attribute 355, 362
 TransferMode property
 of BasicHttpBinding binding 490
 of binding configurations 487–488
 Buffered setting 382
 transport channels 47, 392, 400
 in channel stack 48
 TransportClientCredentialType property 136, 148, 150
 Transport Layer Security (TLS) 125, 127
 transport-level encryption 170
 transport-level security 125, 381, 391, 444, 529
 advantages and disadvantages 126
 for BasicHttpBinding binding
 for WCF client 136–137
 for WCF service 136–137
 implementing for testing 654
 message queue support for 460
 transport protocol 46
 TransportSecurityBindingElement class 404
 TransportWithMessageCredential mode 128, 143
 Trusted People store 186
 importing certificates to 188
 TrustedStoreLocation property 187
 trust relationships between separate domains 124
 trustworthiness
 of client applications 165
 of client certificates, verifying 186–187
 TryCatch activity 323
 Catches section 320
 types of parameters
 changing 218, 223

U

udpAnnouncementEndpoint endpoint 501
 udpDiscoveryEndpoint endpoint 492, 494
 uninstalling Windows service 90
 Universal Description, Discovery, and Integration (UDDI) 656
 UnknownMessageReceived event 57, 118

“Unreachable code detected” warning 191
 “An unsecured or incorrectly secured fault as received from the other party” message 180
 unsubscribe operations in ProductsService service 615–618
 UpdateCustomer method 572, 573–574, 577
 UpdateProductDetails method
 adding implementation 234
 Update Service Reference feature 33
 UriTemplate property 552, 571
 “URL reservation successfully added” message 141
 user account, adding to group 424
 User Account Control dialog box 39
 user application
 hosting service in 54–57
 hosting WCF Service in 84
 username Properties dialog box 424
 user names 625
 usernames 182
 Active Directory query to verify 123
 Windows Integrated Security and 151
 UserName token 172
 users. *See also* authentication
 authorization 152–160
 concurrent, and data changes 353
 creating certificates to identify 188
 displaying name when calling operation in WCF service 146–147
 for WCF service, defining 174–179
 investigating identifiers authenticated with certificates 190–191
 modifying WCF client to supply credentials 148–150
 service authentication of 123
 updating information in SQL Server database 192–195
 using statement 19, 20, 33
 in ShoppingCartService service 361

V

Validate method 411, 412
 validating users with Windows CardSpace 627
 validation of parameters 555

- variables, setting type 315
- VeriSign 138
- versioning
 - and data contract compatibility 238–242
 - services 211–222
- Version Tolerant Serialization 238
- Visual Basic
 - for creating DatabaseFault object 321
 - for workflow activities 299
- Visual Studio 3
 - for building WCF service 9–30
 - installing ASP.NET for .NET Framework 50
 - running as administrator 39
 - Web deployment Tool 326
- void methods, one-way methods as 437
- volatile messages 457
- vulnerability of messages on network 379

W

- WaitForAsyncResult utility method 512
- WCF Data Services 579–598
 - client application
 - exception handlers 597–598
 - client library 587
 - consuming in client application 587–595
 - for modifying data 595–597
- WCF Data Services template 547
- WCF framework
 - flexibility 48
 - predefined bindings 77–80
- WCFPersistence database 529
- WCF runtime
 - acknowledgment message from 386
 - built-in channels 47
 - client application connection to service 419–426
 - determination of certificate validity 141
 - endpoint default values 68
 - exception for exceeded message size 490
 - exception thrown by 272
 - limits on array length in messages 485
 - mappings for converting between XML and .NET Framework types 37
 - round-tripping by 239–242

- selective control of service instance deactivation 270
- single channel stack created by 525
- WCF service startup and 204
- WCF service
 - adding HTTP endpoint 72–74
 - adding method 212–215
 - adding operation 218–222
 - adding reference 19
 - asynchronous operations 447–457
 - building 9–30
 - building application to host 58–76
 - client application connection settings 32
 - code to start and stop 65–67
 - COM+ application as 657–668
 - configuring
 - activation and binding 171–174
 - host environment to support TCP 51–52
 - to send exception details 112–113
 - and testing 24–30
 - to use ASP.NET Role Provider and ASP.NET Membership provider 179–180
 - to use WSDualHttpBinding binding 613–614
 - configuring COM+ application as 661–665
 - creating 13–16
 - defining data contract for 16–17
 - deploying in IIS 39–41
 - without .svc file 29
 - disconnecting from 36
 - displaying messages in 235
 - entity model 10–13
 - exporting certificate 198–199
 - hosting
 - with ASP.NET Web site 170
 - in user application 54–57
 - with Windows application 63
 - with Windows Process Activation Service 49–54
 - in Windows service 84–90
 - how it works 45–49
 - implementing 18–24
 - importing code into IIS Web site 170–171

- message queuing use in 458–460
 - modifying
 - to throw SOAP faults 94–96
 - to throw strongly-typed SOAP faults 101–103
 - to throw unanticipated exception 111
 - in ProductsWorkflowService project 309
 - replacing ASP.NET Web service with 651–654
 - response time and throughput 467
 - sequencing operations in 271–294
 - service contract 18, 203
 - testing 182–185, 207–211
 - with Internet Explorer 27–30
 - timeout 383
 - to support ASP.NET client 647–656
 - transactions 371–372
 - verifying client application authentication 200–201
- WCF Service Configuration Editor 68, 663
- and binding configuration 127
- WCF service contract 47
- WCF Service Library 58
- testing with WcfTestClient application 59–63
- WCF Service Model 399–419, 524
- behaviors 401–403
 - composing channels into bindings 403–408
 - inspecting messages 408–419
 - services and channels 400–401
- WCF service wrapper, for COM+ components 657
- WcfSvcHost utility 59
- displaying 61
- WcfTestClient application 62
- testing WCF Service Library with 59–63
- WCF Test Client window
- displaying 62
- Web applications 3
- creating 169
- Web browser, for testing ProductsSales REST Web service 561–565
- Web.config file 25–26, 649
- <connectionStrings> element 27
 - <protocolMapping> section 83
 - <serviceHostingEnvironment> element 173
 - <system.serviceModel> section 26
 - <system.Web> section
 - <authentication> element 179
 - <roleManager> element 179
- Web deployment Tool 326
- WebFaultException exception 555, 573, 575
- WebGet attribute 552, 570
- WebGet send response messages formatting 552
- WebHttpBehavior endpoint behavior 559
- WebHttpBinding binding 80, 360
- and sessions 278
- WebInvoke attribute 570, 571
- Web resources
- custom AuthorizationManager 635
 - on custom channels 400
 - Custom Demux sample 527
 - data contracts, versioning strategy 231
 - FindPrivateKey utility 197
 - on JSON 6
 - Kerberos protocol 123
 - makecert utility 138
 - on Microsoft Authorization Manager 124
 - OASIS security standards 7
 - on peer-to-peer applications 599
 - public key cryptography 125
 - on Security Token Service 642
 - on SOAP 5
 - Version Tolerant Serialization 238
 - on Windows Azure 9
 - on Windows CardSpace 642
 - on WSDL 5
 - WS-I Basic Profile specification 656
 - XML.org 4
- Web service environment
- maintaining state 243
- WebServiceHost class 559
- Web services 3, 49
- REST model and 548
 - sending and receiving requests 5
 - technical standards and 78

- Web Services Atomic Transaction (WS-AtomicTransaction) specification 351–352
- Web Services Description Language (WSDL)
 - document 5
 - schema 29
- Web Services Enhancements (WSE) 7, 648
- Web sites
 - default protocol bindings for 51
 - user access to 175
- well-known address, for discovery intermediary service 507
- While activity 337, 339
- Windows 7, “There is a problem with the website’s security certificate” error 174
- Windows application, creating to host WCF service 63
- Windows Application Event log 89
- Windows authentication
 - BasicHttpBinding binding configuration to use 150–151
- Windows Azure 8
- Windows CardSpace 626–646
 - identity provider for 641–643
 - information cards for 627, 628
 - self-issued cards for 629
 - validating users with 627
- Windows CardSpace console 637–638
- Windows Communication Foundation Service-Oriented Architecture and 7–9, 42–43
- Windows domain, security implementation 127–162
- Windows environment, authentication in 123–124
- Windows Features dialog box 50
- Windows Firewall 72
- Windows host application
 - configuring 67
- Windows host environment
 - testing 70
- Windows Identity Foundation (WIF) 646
- Windows Integrated Security 124, 143, 262
 - and user names and password transmission 151
- Windows operating system
 - and SOA platform 8
- WindowsPrincipal object 157
- Windows Process Activation Service (WAS)
 - installing and configuring 50
 - WCF service hosting with 49–54
- Windows Security Alert message box 97
- Windows Server AppFabric 54–55, 347
 - and durable service 279
- Windows service
 - creating installer 88
 - hosting WCF service in 84–90
 - installing 88–89
 - starting and stopping 85–87
 - testing 89–90
 - uninstalling 90
- WindowsStreamSecurityBindingElement class 404
- Windows Token Role Provider 124
 - configuring WCF service to use 153–154
- Windows users, authentication of 146–152
- workflow
 - as durable service 347–350
 - for ShoppingCartService service 338–342
- workflow activities
 - Visual Basic syntax for 299
- WorkflowInvoker class 329
- workflow service
 - client application for 313–317
 - creating 313–316
 - testing 317
 - ConfigurationName property 298
 - correlating request and reply messages
 - in instance 335–336
 - debugging 312
 - handling faults 317–325
 - hosting
 - in custom application 328–332
 - in IIS 325–328
 - implementing 296–312
 - implementing common message patterns 332–337
 - passing information into activity 303
 - session management 337–350
 - state maintenance 337–350
 - transactions in 373–377

- WorkflowServiceHost class 328–329
- WorkflowServiceHost object, creating 330
- Workflow Toolbox
 - Messaging section 333–335
- World Wide Web 3
 - REST model and 548
- WriteLine statement 315
 - for tracing 468
- ws2007FederationHttpBinding binding
 - 629
- WS2007FederationHttpBinding binding 80
 - and sessions 278
- WS2007HttpBinding binding 79, 171, 206, 421, 444, 478, 490
 - and client callbacks 606
 - configuring to require Username authentication 150
 - encryption algorithm 145
 - and sessions 278
 - SOAP messaging format for 429
 - WCF client configuration to connect to 144
 - WCF service configuration with endpoint based on 143
- WS2007HttpBinding class 421
- WS2007HttpBinding_IProductsService client endpoint 492
- WS2007HttpBinding_IShoppingCartService endpoint 254
- wsatConfig.exe utility 370
- WS-AtomicTransaction protocol
 - implementing 369–371
- WS-Discovery specification 491
 - versions 494
- WSDL. *See* Web Services Description Language (WSDL)
- WSDL (Web Services Description Language)
 - disabling metadata publishing 561
- WSDualHttpBinding binding 79, 478, 606
 - and reliable messaging 389
 - and sessions 278
 - WCF Service configuration to use 613–614
- WS-Eventing specification 621–622
- WSFederationHttpBinding binding 80, 478
 - and reliable messaging 389
 - and sessions 278
- WSHttpBinding binding 79, 382
 - and client callbacks 606
 - and reliable messaging 389
 - and sessions 278
- WSHttpContextBinding binding 79, 288, 536
 - and sessions 278
- WS-I Basic Profile 78, 656
- WS-Policy framework 43
- WS-ReliableMessaging protocol 78, 379, 380, 381
 - bindings and 389
 - message organization in conversation 384
- WS-SecureConversation specification 395
- WS-Security specification 78, 126
- WS-* specifications
 - ASP.NET Web services and 648
- WS-* Specifications 78
- WS-Transactions specification 78

X

- XAML files, reading 330
- XamlServices class, Load method 330
- .xamlx file extension 297
- XmlDictionaryReader object 486
- XML (eXtensible Markup Language) 4, 46, 548
 - for SOAP messages 476
- XML files
 - Notepad to view 268
 - and privacy 269
- XML schema 4
 - for messages to and from Web service 37
- XML serializer 263
- XmlSerializer object 264, 268

About the Author

John Sharp is a Principal Technologist at Content Master Ltd, a technical authoring company based in the United Kingdom. There he researches and develops technical content for technical training courses, seminars, and white papers. Throughout his development career, John has been active in training, developing, and delivering courses. He has conducted training on subjects ranging from UNIX Systems Programming, to SQL Server Administration, to Enterprise Java Development.

John is deeply involved with .NET development, writing courses, building tutorials, and delivering conference presentations covering Visual C#, WCF, SQL Server, Visual J#, ASP.NET, and Windows Server AppFabric. Apart from *Windows Communication Foundation Step By Step*, John has also authored five editions of *Microsoft Visual C# Step By Step*, and *Microsoft Visual J# .NET*, all published by Microsoft Press.