

THE PRENTICE HALL SERVICE-ORIENTED COMPUTING SERIES FROM THOMAS ERL



"Explaining the intersection of these two worlds—service-orientation and .NET technologies—is exactly what this book does. Its team of specialist authors provides a concrete, usable guide to this combination, ranging from the fundamentals of service-orientation to the more rarified air of .NET services in the cloud and beyond. If you're creating service-oriented software on the Microsoft platform—that is, if you're a serious .NET developer—mastering these ideas is a must."

From the Foreword by David Chappell, Chappell & Associates

SOA with .NET & Windows Azure™

Realizing Service-Oriented Computing with the Microsoft Platform

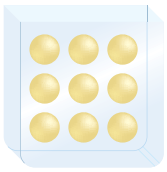
*Edited and Co-Authored by Thomas Erl,
World's Top-Selling SOA Author*

*Forewords by
S. Somasegar
David Chappell*

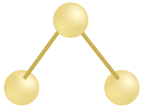
David Chou, John deVadoss, Nitin Gandhi, Hanu Kommapalati,
Brian Loesgen, Christoph Shittko, Herbjörn Wilhelmsen, Mickie Williams

With contributions from Scott Golightly, Daryl Hogan, Jeff King, Scott Seely
With additional contributions by members of the Microsoft Windows Azure and AppFabric teams

PRENTICE
HALL



service
inventory



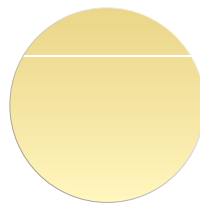
service
composition



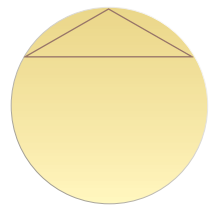
service
(labeled)



service
layer



service
(chorded circle notation)



service accessed
via a uniform interface
(chorded circle notation)



component
or program



decoupled
service contract



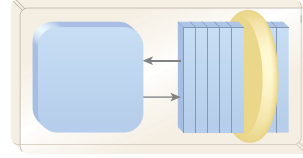
decoupled service
contract accessed
via a uniform
interface



service
agent



firewall



Web service with
service contract



component with
service contract



WSDL
definition



XML Schema
definition



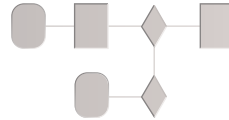
WS-Policy
definition



general machine
processable
document



human
readable
document



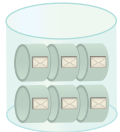
business process
logic



message



security element
or locked resource



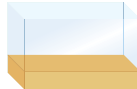
message
queue



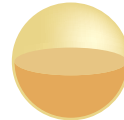
repository
or registry



actively
processing



state data
in memory



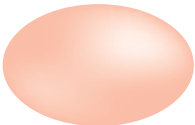
service with state data
(stateful service)



repository with
state data



grid service



zone or
region



conflict
symbol



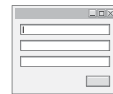
transition
arrow



human



client
workstation



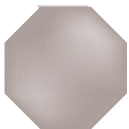
user
interface



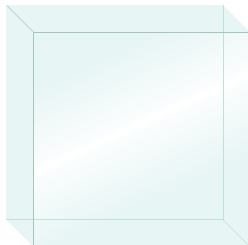
mobile
device



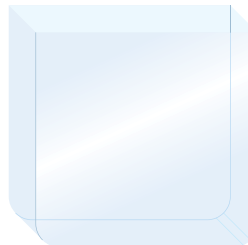
product
or system



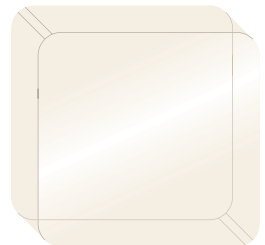
symbols used in conceptual
relationship diagrams



general physical
boundary



service inventory
boundary



service
boundary

Standardized Service Contract (693)

"Services within the same service inventory are in compliance with the same contract design standards."

Service Loose Coupling (695)

"Service contracts impose low consumer coupling requirements and are themselves decoupled from their surrounding environment."

Service Abstraction (696)

"Service contracts only contain essential information and information about services is limited to what is published in service contracts."

Service Reusability (697)

"Services contain and express agnostic logic and can be positioned as reusable enterprise resources."

Service Autonomy (699)

"Services exercise a high level of control over their underlying runtime execution environment."

Service Statelessness (700)

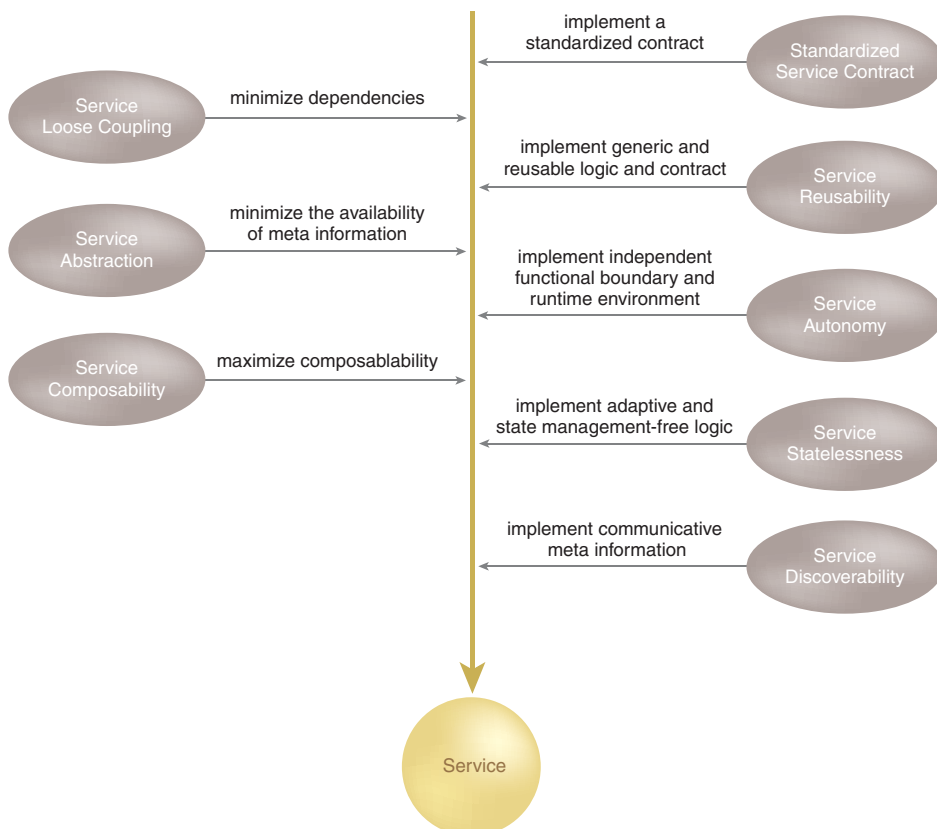
"Services minimize resource consumption by deferring the management of state information when necessary."

Service Discoverability (702)

"Services are supplemented with communicative meta data by which they can be effectively discovered and interpreted."

Service Composability (704)

"Services are effective composition participants, regardless of the size and complexity of the composition."





Agnostic Capability [709] How can multi-purpose service logic be made effectively consumable and composable?



Agnostic Context [710] How can multi-purpose service logic be positioned as an effective enterprise resource?



Agnostic Sub-Controller [711] How can agnostic, cross-entity composition logic be separated, reused, and governed independently?



Asynchronous Queuing [712] How can a service and its consumers accommodate isolated failures and avoid unnecessarily locking resources?



Atomic Service Transaction [713] How can a transaction with rollback capability be propagated across messaging-based services?



Brokered Authentication [714] How can a service efficiently verify consumer credentials if the consumer and service do not trust each other or if the consumer requires access to multiple services?



Canonical Expression [715] How can service contracts be consistently understood and interpreted?



Canonical Protocol [716] How can services be designed to avoid protocol bridging?



Canonical Resources [717] How can unnecessary infrastructure resource disparity be avoided?



Canonical Schema [718] How can services be designed to avoid data model transformation?

Canonical Schema Bus [719]



Canonical Versioning [720] How can service contracts within the same service inventory be versioned with minimal impact?



Capability Composition [721] How can a service capability solve a problem that requires logic outside of the service boundary?



Capability Recomposition [722] How can the same capability be used to help solve multiple problems?



Compatible Change [723] How can a service contract be modified without impacting consumers?



Compensating Service Transaction [724] How can composition runtime exceptions be consistently accommodated without requiring services to lock resources?



Composition Autonomy [725] How can compositions be implemented to minimize loss of autonomy?



Concurrent Contracts [726] How can a service facilitate multi-consumer coupling requirements and abstraction concerns at the same time?



Contract Centralization [727] How can direct consumer-to-implementation coupling be avoided?



Contract Denormalization [728] How can a service contract facilitate consumer programs with differing data exchange requirements?



Cross-Domain Utility Layer [729] How can redundant utility logic be avoided across domain service inventories?



Data Confidentiality [730] How can data within a message be protected so that it is not disclosed to unintended recipients while in transit?



Data Format Transformation [731] How can services interact with programs that communicate with different data formats?



Data Model Transformation [732] How can services interoperate when using different data models for the same type of data?



Data Origin Authentication [733] How can a service verify that a message originates from a known sender and that the message has not been tampered with in transit?



Decomposed Capability [734] How can a service be designed to minimize the chances of capability logic deconstruction?



Decoupled Contract [735] How can a service express its capabilities independently of its implementation?

(pattern list continued on inside back cover)

This page intentionally left blank

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

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

For sales outside the United States please contact:

International Sales
international@pearson.com

Visit us on the Web: informit.com/ph

The Library of Congress Cataloging-in-Publication Data is on file.

Copyright © 2010 SOA Systems Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax (617) 671 3447

ISBN-13: 978-0-13-158231-6

ISBN-10: 0-13-158231-3

Text printed in the United States on recycled paper at Edward Brothers in Ann Arbor, Michigan.

First printing May 2010

Editor-in-Chief

Mark L. Taub

Development Editor

Christina Erl-Daniels

Managing Editor

Kristy Hart

Project Editor

Betsy Harris

Senior Indexer

Cheryl Lenser

Proofreaders

Williams Woods Publishing
Amy Chou

Publishing Coordinator

Kim Boedigheimer

Cover Designers

Thomas Erl
Ivana Lee

Compositor

Bumpy Design

Photos

Thomas Erl

Diagram Designer

Christina Erl-Daniels

Contents

Foreword by S. Somasegar xxxi

Foreword by David Chappell xxxiii

Acknowledgments xxxv

CHAPTER 1: Introduction 1

1.1 About this Book 2

1.2 Objectives of this Book 3

1.3 Who this Book is For 4

1.4 What this Book Does Not Cover 4

1.5 Prerequisite Reading 4

1.6 How this Book is Organized 6

Part I: Fundamentals 7

Chapter 3: SOA Fundamentals 7

Chapter 4: A Brief History of Legacy .NET Distributed Technologies . . 7

Chapter 5: WCF Services 7

Chapter 6: WCF Extensions 7

Chapter 7: .NET Enterprise Services Technologies 7

Chapter 8: Cloud Services with Windows Azure 8

Part II: Services and Service Composition 8

*Chapter 9: Service-Orientation with .NET Part I:
Service Contracts and Interoperability 8*

*Chapter 10: Service-Orientation with .NET Part II:
Coupling, Abstraction, and Discoverability 8*

*Chapter 11: Service-Orientation with .NET Part III:
Reusability and Agnostic Service Models 8*

*Chapter 12: Service-Orientation with .NET Part IV:
Service Composition and Orchestration Basics 9*

Chapter 13: Orchestration Patterns with WF 9

Chapter 14: Orchestration Patterns with BizTalk Server 9

| | |
|---|-----------|
| Part III: Infrastructure and Architecture | 9 |
| <i>Chapter 15: Enterprise Service Bus with BizTalk Server and Windows Azure</i> | <i>9</i> |
| <i>Chapter 16: Windows Azure Platform AppFabric Service Bus.</i> | <i>10</i> |
| <i>Chapter 17: SOA Security with .NET and Windows Azure</i> | <i>10</i> |
| <i>Chapter 18: Service-Oriented Presentation Layers with .NET</i> | <i>10</i> |
| <i>Chapter 19: Service Performance Optimization</i> | <i>10</i> |
| <i>Chapter 20: SOA Metrics with BAM</i> | <i>10</i> |
| Part IV: Appendices | 10 |
| <i>Appendix A: Case Study Conclusion</i> | <i>10</i> |
| <i>Appendix B: Industry Standards Reference.</i> | <i>11</i> |
| <i>Appendix C: Service-Oriented Principles Reference</i> | <i>11</i> |
| <i>Appendix D: SOA Design Patterns Reference</i> | <i>11</i> |
| <i>Appendix E: The Annotated SOA Manifesto.</i> | <i>11</i> |
| <i>Appendix F: Additional Resources</i> | <i>11</i> |
| 1.7 How Principles and Patterns are Used in this Book | 11 |
| Sources | 11 |
| Reference Notation | 12 |
| 1.8 Symbols, Figures, and Style Conventions. | 13 |
| Symbol Legend | 13 |
| How Color is Used | 13 |
| Additional Information | 13 |
| Updates, Errata, and Resources (www.soabooks.com) | 13 |
| Master Glossary (www.soaglossary.com) | 13 |
| Referenced Specifications (www.soaspecs.com) | 13 |
| SOASchool.com™ SOA Certified Professional (SOACP) | 14 |
| The <i>SOA Magazine</i> (www.soamag.com) | 14 |
| Notification Service | 14 |

CHAPTER 2: Case Study Background 15

| | |
|---|----|
| 2.1 How Case Studies Are Used. | 16 |
| 2.2 Case Study Background #1: Standard Mold | 16 |
| History | 16 |
| Technical Infrastructure. | 16 |
| Business Goals and Obstacles. | 17 |

| | |
|---|----|
| 2.3 Case Study Background #2: Superior Stamping | 18 |
| History | 18 |
| Technical Infrastructure | 18 |
| Business Goals and Obstacles | 19 |

PART I: FUNDAMENTALS

CHAPTER 3: SOA Fundamentals 23

| | |
|--|----|
| 3.1 Basic SOA Terminology | 24 |
| Service-Oriented Computing | 25 |
| Service-Oriented Architecture (SOA) | 27 |
| Services | 28 |
| <i>Services as Components</i> | 29 |
| <i>Services as Web Services</i> | 30 |
| <i>Services as REST Services</i> | 31 |
| Service Models | 31 |
| <i>Agnostic Logic and Non-Agnostic Logic</i> | 32 |
| Service Composition | 33 |
| Service Inventory | 34 |
| Service-Oriented Analysis | 34 |
| Service Candidate | 35 |
| Service-Oriented Design | 35 |
| Service Contract | 36 |
| Service-Related Granularity | 37 |
| SOA Design Patterns | 38 |
| 3.2 Service-Oriented Computing Goals | 40 |
| Increased Intrinsic Interoperability | 40 |
| Increased Federation | 40 |
| Increased Vendor Diversification Options | 40 |
| Increased Business and Technology Domain Alignment | 41 |
| 3.3 Further Reading | 41 |

| | |
|---|-----------|
| CHAPTER 4: A Brief History of Legacy .NET Distributed Technologies | 43 |
| 4.1 Distributed Computing 101 | 44 |
| Client-Server | 44 |
| Distributed Architecture | 45 |
| Service-Oriented Architecture | 47 |
| 4.2 .NET Enterprise Services | 48 |
| It All Began with COM (and DCOM) | 48 |
| COM+ Services | 49 |
| .NET Assemblies | 51 |
| Distributed Transaction Coordinator | 51 |
| .NET Enterprise Services and Service-Orientation | 53 |
| 4.3 .NET Remoting | 54 |
| .NET Remoting Architecture | 54 |
| <i>Serializable Classes</i> | 56 |
| <i>Remotable Classes</i> | 56 |
| <i>Ordinary Classes</i> | 56 |
| Hosting .NET Remoting Components | 56 |
| <i>Windows Service</i> | 56 |
| <i>IIS Hosting Under ASP.NET</i> | 57 |
| <i>Hosting a .NET Remoting Component in a Console Application</i> | 57 |
| <i>.NET COM+ Services</i> | 57 |
| .NET Remoting Configurations | 57 |
| <i>Activation Types</i> | 58 |
| <i>Message Formats</i> | 60 |
| <i>Communication Protocols</i> | 60 |
| <i>Object Lifetime Management</i> | 61 |
| .NET Remoting and Service-Orientation | 61 |
| 4.4 Microsoft Messaging Queue (MSMQ). | 63 |
| The Queues | 64 |
| Sending and Receiving Messages | 65 |
| MSMQ and Service-Orientation | 66 |
| 4.5 System.Transactions | 67 |
| Distributed Resource Transactions | 67 |
| Explicit and Implicit Programming Models | 68 |
| Ambient Transactions | 69 |

| | |
|---|----|
| 4.6 Web Services (ASMX and WSE) | 70 |
| XML Web Services (ASMX) | 71 |
| <i>The WebService Attribute</i> | 71 |
| <i>The WebMethod Attribute</i> | 72 |
| Web Service Enhancements (WSE) | 73 |
| 4.7 REST Service Processing with IHttpHandler | 74 |

CHAPTER 5: WCF Services 75

| | |
|--|-----|
| 5.1 Overview | 76 |
| 5.2 Service Contracts with WCF | 78 |
| WCF Terminology | 78 |
| <i>WCF Service Contract</i> | 78 |
| <i>Interface Contract</i> | 78 |
| <i>Operation Contract</i> | 78 |
| <i>Data Contract</i> | 78 |
| <i>Message Contract</i> | 79 |
| <i>Service Endpoint</i> | 79 |
| The ServiceContract and OperationContract Attributes | 79 |
| Data Models and the DataContract Attribute | 82 |
| Messaging and the MessageContract Attribute | 83 |
| Service Endpoints and the endpoint Element | 86 |
| <i>Address</i> | 88 |
| <i>Bindings</i> | 89 |
| <i>Contract</i> | 92 |
| REST Service Classes and Attributes | 92 |
| <i>The WebGet Attribute</i> | 93 |
| <i>The WebInvoke Attribute</i> | 95 |
| <i>WCF UriTemplate Attribute</i> | 96 |
| Faults and the FaultContract Attribute | 98 |
| MEX Endpoints | 100 |
| Versioning Considerations | 102 |
| 5.3 Service Implementation with WCF | 104 |
| Behaviors | 104 |
| Instancing | 105 |
| A Sample Implementation | 106 |

| | |
|---|-----|
| 5.4 Service Hosting with WCF | 108 |
| Self-Hosted Services | 110 |
| Managed Windows Services | 112 |
| IIS Process Boundary | 113 |
| Windows Activation Services (WAS) | 114 |
| Hosting REST Services | 115 |
| 5.5 Service Consumers with WCF | 116 |
| Using the Service Metadata Tool | 117 |
| Writing the Proxy Class for a Service | 118 |
| Using the ChannelFactory Class | 119 |

CHAPTER 6: WCF Extensions 121

| | |
|--|-----|
| 6.1 WCF Security | 122 |
| Security Modes | 123 |
| Authorization | 125 |
| Federated Identity | 126 |
| 6.2 WCF Transactions | 127 |
| Operation Attributes for Transactions | 127 |
| <i>TransactionScopeRequired</i> | 128 |
| <i>TransactionAutoComplete</i> | 128 |
| <i>TransactionFlow</i> | 128 |
| Service Attributes for Transactions | 129 |
| <i>TransactionIsolationLevel</i> | 129 |
| <i>TransactionAutoCompleteOnSessionClose</i> | 130 |
| <i>TransactionTimeout</i> | 130 |
| Durable Services | 131 |
| 6.3 WCF Router | 132 |
| The RoutingService Class | 133 |
| Routing Contracts | 134 |
| Routing Configuration | 135 |
| <i>Step 1: Define Endpoints</i> | 135 |
| <i>Step 2: Configure Service Behavior</i> | 136 |
| <i>Step 3: Enumerate Target Endpoints</i> | 136 |
| <i>Step 4: Define Message Filters</i> | 137 |
| <i>Step 5: Create a Filter Table</i> | 138 |
| Fault Tolerance | 139 |

| | |
|--|-----|
| 6.4 WCF Discovery | 140 |
| Discovery Modes | 141 |
| Locating a Service Ad Hoc | 143 |
| Sending and Receiving Service Announcements. | 144 |
| Discovery Proxies for Managed Discovery | 146 |
| <i>Discovering from a Discovery Proxy.</i> | 146 |
| Implicit Service Discovery | 147 |
| 6.5 WCF Extensibility. | 148 |
| WCF Layers | 149 |
| Layered Extensibility | 149 |
| Channel Layer Extensibility | 150 |
| 6.6 WCF Management Tools | 151 |
| Administration | 151 |
| Troubleshooting | 151 |
| Logging Messages | 153 |

CHAPTER 7: .NET Enterprise Services Technologies 155

| | |
|---|-----|
| 7.1 SQL Server | 156 |
| Native XML Web Services Support. | 157 |
| Service Broker (SSB). | 160 |
| Query Notification | 165 |
| XML Support in SQL Server | 165 |
| 7.2 Windows Workflow Foundation (WF). | 166 |
| WF Architecture. | 167 |
| Workflows | 168 |
| <i>Sequential Workflows</i> | 169 |
| <i>State Machine Workflows</i> | 169 |
| Workflow Designer | 169 |
| Workflow Persistence (with WF) | 170 |
| Communicating with the Host Container. | 171 |
| Activities | 172 |
| Workflow Runtime Environment. | 175 |
| WF Programming Model | 176 |
| Passing Parameters into a Workflow Instance | 178 |

| | |
|---|-----|
| Returning Parameters from a Workflow Instance | 178 |
| Workflow-Enabled Services. | 179 |
| Versioning Orchestrations | 180 |
| WF Extensibility | 180 |
| Business Rules | 180 |
| 7.3 Application Blocks and Software Factories | 181 |
| Application Blocks | 182 |
| Software Factories. | 184 |
| <i>Guidance Toolkits</i> | 184 |
| <i>Web Services Software Factory</i> | 184 |
| 7.4 Windows Server AppFabric | 187 |
| Configurable Hosting Environment | 188 |
| Workflow Persistence (with AppFabric) | 189 |
| In-Memory Application Cache Platform | 190 |
| Manageability Extensions | 192 |
| Application Server Event Collector | 192 |
| 7.5 BizTalk Server | 193 |
| BizTalk Server Architecture | 194 |
| Messaging | 196 |
| <i>Pipelines</i> | 197 |
| <i>Pipeline Components</i> | 198 |
| <i>Ports and Locations</i> | 199 |
| Adapters | 199 |
| Context Properties. | 200 |
| Itineraries | 201 |
| Unified Exception Management | 202 |

CHAPTER 8: Cloud Services with Windows Azure 205

| | |
|--|-----|
| 8.1 Cloud Computing 101. | 206 |
| Cloud Deployment Models | 208 |
| <i>Public Cloud</i> | 208 |
| <i>Private Cloud</i> | 208 |
| <i>Community Cloud</i> | 209 |
| <i>Other Deployment Models</i> | 209 |
| <i>The Intercloud (Cloud of Clouds)</i> | 209 |
| <i>Deployment Models and Windows Azure</i> | 210 |

| | |
|--|-----|
| Service Delivery Models | 210 |
| <i>Infrastructure-as-a-Service (IaaS)</i> | 210 |
| <i>Platform-as-a-Service (PaaS)</i> | 211 |
| <i>Software-as-a-Service (SaaS)</i> | 211 |
| <i>Other Delivery Models</i> | 211 |
| <i>IaaS vs. PaaS</i> | 211 |
| 8.2 Windows Azure Platform Overview | 213 |
| Windows Azure (Application Container) | 216 |
| SQL Azure | 217 |
| Windows Azure Platform AppFabric | 218 |
| 8.3 Windows Azure Roles | 219 |
| <i>Web Roles and Worker Roles</i> | 220 |
| <i>Virtual Machines</i> | 220 |
| <i>Input Endpoints</i> | 221 |
| <i>Inter-Role Communication</i> | 222 |
| 8.4 Hello World in Windows Azure | 223 |
| 1. Create a Cloud Service Project | 224 |
| 2. Choose an ASP.NET Web Role | 224 |
| 3. Create the Solution | 225 |
| 4. Instantiate the Service | 226 |
| 8.5 A Web Service in Windows Azure | 227 |
| 1. Create a Host Service and Storage Service | 233 |
| 2. Create and Deploy a Service Package | 233 |
| 3. Promote the Service to Production | 234 |
| 8.6 A REST Service in Windows Azure | 235 |
| REST Service Addressing | 235 |
| Creating a Windows Azure REST Service | 236 |
| 8.7 Windows Azure Storage | 239 |
| Tables | 240 |
| <i>Entities and Properties</i> | 240 |
| <i>Data Access</i> | 241 |
| Queues | 241 |
| Blobs | 242 |
| <i>Block Blobs</i> | 242 |
| <i>Page Blobs</i> | 243 |
| Windows Azure Drive | 243 |

PART II: SERVICES AND SERVICE COMPOSITION

CHAPTER 9: Service-Orientation with .NET Part I: Service Contracts and Interoperability 247

| | |
|--|-----|
| 9.1 Standardized Service Contract. | 250 |
| Contract-First. | 250 |
| 1. <i>Create or Reuse Data Contract</i> | 251 |
| 2. <i>Create Message Contract</i> | 251 |
| 3. <i>Create Interface Contract</i> | 252 |
| Standardized Service Contract and Patterns | 252 |
| 9.2 Canonical Schema | 253 |
| Creating Schemas with Visual Studio | 254 |
| Generating .NET Types | 258 |
| Using the DataContract Library | 264 |
| 9.3 Data Model Transformation | 267 |
| Object-to-Object | 269 |
| LINQ-to-XML | 271 |
| XSLT Transformation | 272 |
| 9.4 Canonical Protocol | 274 |
| Web Service | 275 |
| REST Service. | 277 |
| Component | 278 |
| Another WCF Option: Named Pipes | 279 |
| Dual Protocols with WCF. | 279 |
| 9.5 Canonical Expression | 280 |
| Service Naming Conventions | 280 |
| Service Capability Naming Conventions. | 281 |

CHAPTER 10: Service-Orientation with .NET Part II: Coupling, Abstraction, and Discoverability. 283

| | |
|--|-----|
| 10.1 Service Loose Coupling | 285 |
| Service Loose Coupling and Patterns. | 286 |

| | |
|---|-----|
| 10.2 Decoupled Contract | 288 |
| WSDL-First | 289 |
| Generating Service Code Using Svcutil | 294 |
| Generating WCF Service Code Using WSCF.blue | 297 |
| Generating ASMX Service Code Using WSCF.classic | 302 |
| 10.3 Service Façade | 304 |
| 10.4 Concurrent Contracts | 307 |
| 10.5 Service Loose Coupling and Service Capability Granularity | 308 |
| 10.6 Service Abstraction | 313 |
| 10.7 Validation Abstraction | 315 |
| 10.8 Exception Shielding | 319 |
| 10.9 Service Discoverability | 321 |
| In-line Documentation | 322 |
| REST and Hypermedia | 323 |
| Service Profiles | 323 |
| 10.10 Metadata Centralization | 325 |

CHAPTER 11: Service-Orientation with .NET Part III: Reusability and Agnostic Service Models 327

| | |
|---|-----|
| 11.1 Service Reusability and the Separation of Concerns . . . | 329 |
| Functional Decomposition | 330 |
| Service Encapsulation | 332 |
| Agnostic Context | 332 |
| Agnostic Capability | 334 |
| Utility Abstraction | 335 |
| Entity Abstraction | 336 |
| The Inventory Analysis Cycle | 337 |
| Additional Design Considerations | 339 |
| 11.2 Case Study Example: Utility Abstraction with a .NET Web Service | 339 |
| 11.3 Case Study Example: Entity Abstraction with a .NET REST Service | 351 |

CHAPTER 12: Service-Orientation with .NET Part IV: Service Composition and Orchestration Basics 369

| | |
|--|-----|
| 12.1 Service Composition 101 | 371 |
| Service-Orientation and Service Composition | 371 |
| Service Composability (PSD) | 373 |
| Capability Composition and Capability Recomposition | 374 |
| <i>Capability Composition</i> | 375 |
| <i>Capability Recomposition</i> | 375 |
| Composition Roles | 377 |
| Service Layers. | 377 |
| Non-Agnostic Context | 379 |
| Process Abstraction and Task Services | 380 |
| 12.2 Orchestration. | 382 |
| Process Abstraction, Process Centralization, and Orchestrated Task Services | 382 |
| <i>Process Centralization and Tools</i> | 384 |
| <i>Process Abstraction and WS-BPEL</i> | 385 |
| State Repository and Compensating Service Transaction | 385 |
| <i>State Repository with .NET</i> | 386 |
| <i>Compensating Service Transaction</i> | 387 |
| Other Patterns | 388 |
| Microsoft Orchestration Platforms: WF and BizTalk Server | 388 |

CHAPTER 13: Orchestration Patterns with WF 393

| | |
|--|-----|
| 13.1 Process Abstraction and Orchestrated Task Services . . | 397 |
| A Brief History of WF Service Contract Support | 397 |
| Publishing WF Workflows as Web Services and Activities | 399 |
| <i>Workflows Published as ASMX Services</i> | 399 |
| <i>Workflows Published via WCF 3.5 Activities</i> | 408 |
| <i>Workflows Published via WCF 4.0 Activities</i> | 410 |
| <i>Workflows Published via ExternalDataExchange Services</i> | 413 |
| <i>WS-I BasicProfile Support</i> | 417 |

| | |
|--|-----|
| Publishing WF Workflows as REST Services | 419 |
| <i>JSON Encoding</i> | 421 |
| <i>Send and Receive Activity Configuration</i> | 422 |
| <i>Orchestrated Task Services with REST and WF 4.0</i> | 423 |
| 13.2 Process Centralization | 425 |
| Centralized Process Maintenance | 425 |
| WS-BPEL Support | 426 |
| 13.3 State Repository | 426 |
| SQL Persistence Service and Scaling Out in WF 3.0 | 429 |
| SQL Persistence Service and Scaling Out in WF 4 | 431 |
| 13.4 Compensating Service Transaction | 434 |
| Creating Compensations | 434 |
| Triggering Compensations | 435 |
| 13.5 Case Study Example | 436 |

CHAPTER 14: Orchestration Patterns with BizTalk Server 441

| | |
|--|-----|
| 14.1 Process Abstraction and Orchestrated Task Services . . | 443 |
| Orchestrated Task Service Contracts | 445 |
| WS-* Support | 447 |
| Case Study Example | 448 |
| 14.2 Process Centralization | 450 |
| Centralized Process Maintenance | 450 |
| WS-BPEL Support | 451 |
| <i>Exporting BizTalk Orchestrations to WS-BPEL</i> | 451 |
| <i>Importing WS-BPEL Processes into BizTalk</i> | 454 |
| 14.3 State Repository | 455 |
| 14.4 Compensating Service Transaction | 456 |
| Case Study Example | 459 |

PART III: INFRASTRUCTURE AND ARCHITECTURE

CHAPTER 15: Enterprise Service Bus with BizTalk Server and Windows Azure 465

| | |
|---|-----|
| 15.1 Microsoft and the ESB. | 466 |
| 15.2 Integration with BizTalk | 467 |
| Application Integration 101 | 467 |
| The BizTalk Hub-Bus Model | 469 |
| 15.3 The ESB Toolkit | 470 |
| Itineraries | 472 |
| <i>Itineraries Types</i> | 474 |
| <i>The Itinerary Lifecycle</i> | 475 |
| Resolvers. | 476 |
| Adapter Providers | 478 |
| <i>WCF-Custom and REST Services</i> | 479 |
| 15.4 Distributed and Scalable ESB Architecture | 480 |
| Configuring for High-Availability | 480 |
| Techniques for Scaling | 481 |
| Distributed ESBs | 482 |
| 15.5 Cloud-Enabling the ESB with Windows Azure | 483 |
| Receiving Messages from Azure's AppFabric Service Bus | 484 |
| Sending Messages to Azure's AppFabric Service Bus. | 485 |
| 15.6 Governance Considerations | 487 |
| SLA Enforcement. | 488 |
| Monitoring | 488 |
| Preparing Project Teams | 489 |
| 15.7 Mapping the Microsoft Platform to the Enterprise Service Bus Pattern. | 490 |

CHAPTER 16: Windows Azure Platform AppFabric Service Bus 493

| | |
|---|-----|
| 16.1 Introducing the Service Bus | 494 |
| Connectivity Fabric | 494 |
| Message Buffers | 496 |
| Service Registry | 497 |
| 16.2 Service Bus and REST | 498 |
| REST-Based Service Design | 498 |
| REST-Based Service Consumer Design | 499 |
| Message Buffers and REST | 499 |
| 16.3 Service Bus Connectivity Models | 499 |
| Eventing | 500 |
| Service Remoting | 501 |
| Tunneling | 501 |
| 16.4 Working with Windows Azure Platform AppFabric Service Bus | 503 |
| Setting up the AppFabric Service Bus | 504 |
| Defining a REST-Based Service Bus Contract | 513 |
| Creating the Service Bus Message Buffer | 514 |

CHAPTER 17: SOA Security with .NET and Windows Azure 517

| | |
|--|-----|
| 17.1 Authentication and Authorization with WCF | 518 |
| Direct and Brokered Authentication | 518 |
| <i>Direct Authentication.</i> | 518 |
| <i>Brokered Authentication.</i> | 519 |
| <i>Authentication Patterns in WCF.</i> | 520 |
| Role-Based Authorization | 520 |
| <i>Authorization Roles in WCF.</i> | 521 |
| <i>Authorizing Operations with Roles.</i> | 523 |
| Claims-Based Authorization | 524 |
| <i>Claims Processing in WCF.</i> | 526 |
| <i>Implementing Claims-Based Authorization.</i> | 527 |
| <i>Access Control in Windows Azure.</i> | 528 |
| <i>Designing Custom Claims.</i> | 529 |

| | |
|--|-----|
| Case Study Example | 530 |
| 17.2 Windows Identity Foundation (WIF) | 533 |
| Digital Identity | 534 |
| <i>The Identity Metasystem</i> | 534 |
| Windows Cardspace | 536 |
| Active Directory Federation Services (ADFS) | 539 |
| WIF Programming Model | 540 |
| <i>WCF Integration</i> | 540 |
| <i>Programming Windows Cardspace</i> | 540 |
| Developing a Relying Party | 541 |
| Developing an Identity Provider | 542 |
| 17.3 Windows Azure Security | 543 |
| Cloud Computing Security 101 | 543 |
| <i>Cross-Domain Access Control</i> | 544 |
| <i>Hybrid Cloud Security</i> | 545 |
| <i>Inter-Organization Service Composition Security</i> | 545 |
| <i>External Identity Providers</i> | 546 |
| <i>Claims-Based Access Control, As-A-Service</i> | 546 |
| Windows Azure Platform AppFabric Access Control Overview | 548 |
| <i>Access Control Step-by-Step</i> | 550 |
| <i>Access Control and REST</i> | 552 |
| Access Control Service Authorization Scenarios | 553 |
| <i>Hybrid Cloud Authorization Model</i> | 553 |
| <i>Public Cloud Authorization Model</i> | 554 |
| <i>Cloud-to-Cloud Authorization Model</i> | 554 |
| Case Study Example | 555 |

CHAPTER 18: Service-Oriented Presentation Layers with .NET 557

| | |
|---|-----|
| 18.1 Windows Presentation Foundation and the Prism Library | 559 |
| Shell | 561 |
| Views | 562 |
| <i>View Discovery versus View Injection</i> | 563 |
| Regions | 563 |

| | |
|---|-----|
| Modules | 565 |
| Shared Services | 566 |
| 18.2 Design Patterns for Presentation Logic. | 567 |
| User Interface Patterns | 567 |
| <i>Composite View [CJP]</i> | 568 |
| <i>Command [DP]</i> | 568 |
| <i>UI Mediator</i> | 568 |
| <i>Separated Presentation</i> | 568 |
| Modularity Patterns | 569 |
| <i>Separated Interface [PEA]</i> | 570 |
| <i>Plug-In [PEA]</i> | 570 |
| <i>Event Aggregator [PEA]</i> | 570 |
| <i>Inversion of Control [DP]</i> | 570 |
| <i>Dependency Injection [PEA]</i> | 570 |
| <i>Service Locator [CJP]</i> | 571 |
| 18.3 A Simple Service-Oriented User Interface | 571 |
| Creating the Project. | 571 |
| Dynamically Loading Modules | 579 |

CHAPTER 19: Service Performance Optimization. . . . 583

| | |
|--|-----|
| 19.1 Overview | 584 |
| Optimization Areas | 585 |
| Service Implementation Processing | 585 |
| Service Framework Processing. | 586 |
| Wire Transmission Processing. | 586 |
| 19.2 Service Performance Optimization Techniques | 586 |
| Caching to Avoid Costly Processing. | 587 |
| <i>Intermediary</i> | 589 |
| <i>Service Container</i> | 589 |
| <i>Service Proxy</i> | 590 |
| <i>Caching Utility Service</i> | 590 |
| <i>Comparing Caching Techniques</i> | 591 |
| Cache Implementation Technologies | 592 |
| Computing Cache Keys | 593 |

| | |
|--|-----|
| Case Study Example | 594 |
| <i>Method 1</i> | 597 |
| <i>Method 2</i> | 598 |
| Caching REST Responses | 599 |
| Monitoring Cache Efficiency | 601 |
| Reducing Resource Contention | 603 |
| Request Throttling | 604 |
| <i>Throttling With WCF</i> | 605 |
| Case Study Example | 606 |
| <i>Request Throttling with BizTalk Server</i> | 607 |
| Coarse-Grained Service Contracts | 608 |
| Case Study Example | 609 |
| Selecting Application Containers | 610 |
| Performance Policies | 612 |
| Case Study Example | 620 |
| REST Service Message Sizes | 621 |
| Hardware Encryption | 622 |
| <i>Transport Encryption</i> | 622 |
| <i>Message Encryption</i> | 623 |
| <i>Custom Encryption Solution</i> | 623 |
| High Performance Transport | 625 |
| Case Study Example | 626 |
| MTOM Encoding | 627 |
| Case Study Example | 628 |
| Performance Considerations for Service Contract Design | 630 |
| Case Study Example | 631 |
| Impact on Service-Oriented Principles | 633 |
| 19.3 Service Composition Performance | |
| Optimization Techniques | 637 |
| Transformation Avoidance and Caching | 637 |
| Asynchronous Interactions | 639 |
| Parallelize Where Possible | 641 |
| <i>Parallel Activity in WF</i> | 641 |
| <i>Parallel Execution in BizTalk Server</i> | 643 |
| <i>Replicator Activity in WF</i> | 644 |

| | |
|--|-----|
| Consider Co-Hosting When Necessary | 645 |
| Compose High Performance Services | 648 |
| Impact on Service-Orientation Principles | 648 |

CHAPTER 20: SOA Metrics with BAM 653

| | |
|--|-----|
| 20.1 SOA Metric Types | 654 |
| 20.2 Introducing BizTalk BAM | 655 |
| BizTalk and BAM | 655 |
| BAM Solution Architecture | 656 |
| The BAM Management Utility | 659 |
| The Tracking Profile Editor (TPE) | 659 |
| Real-Time vs Scheduled Aggregations | 660 |
| 20.3 Activities and Views | 661 |
| Roles-based Views for Service Governance | 662 |
| Creating Views | 663 |
| 20.4 BAM APIs | 665 |
| Event Streams | 665 |
| <i>DirectEventStream (DES)</i> | 665 |
| <i>BufferedEventStream (BES)</i> | 665 |
| <i>OrchestrationEventStream (OES)</i> | 666 |
| <i>IPipelineContext Interface</i> | 666 |
| Abstracted APIs for Service Metrics | 666 |
| Metrics for Service Compositions | 669 |
| WCF and WF Interceptors | 670 |
| Notifications | 670 |
| Rapid Prototyping | 671 |
| 20.5 Managing BAM | 672 |
| Database Outages | 672 |
| Security | 672 |
| Scripting Deployment | 673 |
| Reporting | 676 |
| Case Study Example | 677 |

PART IV: APPENDICES

APPENDIX A: Case Study Conclusion 685

APPENDIX B: Industry Standards Reference 687

APPENDIX C: Service-Orientation Principles Reference 691

APPENDIX D: SOA Design Patterns Reference 707

APPENDIX E: The Annotated SOA Manifesto 795

The Annotated SOA Manifesto 796

APPENDIX F: Additional Resources 809

| | |
|---|-----|
| Consuming Services with WCF | 811 |
| Introduction | 811 |
| Cleaning Up Resources | 812 |
| The Proper Disposal and Closing of an ICommunicationObject | 812 |
| The ICommunicationObject.Close() Method | 812 |
| The ICommunicationObject.Abort() Method | 814 |
| Abort() versus Close() | 814 |
| IDisposable for Cleaning Up Resources | 814 |
| IDisposable and Its Relation to ClientBase and ChannelFactory | 815 |
| Cleaning Up Resources with the Using Block | 816 |
| Cleaning Up Resources with the Try-Catch-Finally-Abort Pattern | 817 |
| Handling Exceptions and Cleaning Up Resources with the Try-Close-Catch-Abort Pattern | 818 |
| Cleaning Up Resources in a Convenient Way | 819 |

| | |
|---|-----|
| How to Handle Connections when Consuming Services | |
| Using WCF | 822 |
| Conclusion | 823 |

About the Authors 825

| | |
|-------------------------------|-----|
| David Chou | 825 |
| John deVadoss | 825 |
| Thomas Erl | 826 |
| Nitin Gandhi | 826 |
| Hanu Kommalapati | 827 |
| Brian Loesgen | 827 |
| Christoph Schittko | 828 |
| Herbjörn Wilhelmsen | 828 |
| Mickey Williams | 828 |

About the Contributors 829

| | |
|---------------------------|-----|
| Scott Golightly | 829 |
| Darryl Hogan | 829 |
| Kris Horrocks | 829 |
| Jeff King | 830 |
| Scott Seely | 830 |

About the Foreword Contributors 831

| | |
|--------------------------|-----|
| David Chappell | 831 |
| S. Somasegar | 831 |

Index 833

This page intentionally left blank

Foreword by S. Somasegar

Within the last decade, service-oriented architecture has moved from infancy to ubiquity, precipitating and, in many ways, enabling the next paradigm of information technology—cloud computing. As the Software + Services model becomes the norm, businesses that embrace SOA will smoothly transition to the cloud, enabling better scaling, availability, and cost efficacy of their services.

Service-oriented architecture is the magic behind many of the Internet-based applications and services that seamlessly integrate information and data from multiple sources into a single experience. The loosely-coupled design of SOA allows developers to take advantage of existing services to build their applications, dynamically adapt to changes in those services, and offer their own services to other application developers. From a developer perspective, it is best to have tools and frameworks that enable you to write **ONLY** the code that you need to write. SOA allows developers to do just that—focus on building the unique parts of their applications by enabling them to reuse existing services others have already written to solve common problems.

Microsoft has long promoted a real-world approach to SOA that focuses on helping organizations create business value by translating business challenges into opportunities. A real-world approach is typically based on rapid, agile iterations of design, implementation, and assessment, resulting in solutions that are able to better track and align to the changing needs of the organization and its business environment.

To developers, service-orientation offers a technology model with the potential for effectively creating and maintaining ‘evolvable’ applications: applications that are better able to change and grow with the business. To the CIO, service-orientation offers strategies and tools for nurturing existing IT assets while simultaneously fostering the development of new capabilities. The ‘rip and replace’ tactic of the past to deal with changing technology and business needs is facing extinction, thanks primarily to the increasing adoption of service-orientation and the pervasive nature of service-oriented architectures. The encapsulation of existing assets behind service-based interfaces provides structured access to legacy systems and applications while facilitating the opportunity for continuous improvement of the underlying business capabilities behind the interface.

However, keep in mind that architecture is a means to an end. The end goal is to create continuing value for your business, and a real-world approach based on proven practices offers a viable map to help get you there. With the emergence of the cloud as an attractive platform for both consumer and business computing, the principles underlying loose-coupling and service-orientation are increasingly relevant beyond the four walls of the data center. Services are more and more being developed and deployed beyond the confines of the firewall.

Microsoft’s innovative new platforms and tools, including Windows Azure and SQL Azure, as well as Windows Azure platform AppFabric, Visual Studio 2010, and .NET Framework 4, enable organizations to extend their service-oriented architectures into the cloud, creating a hybrid Software + Services model. The Microsoft platforms and tools provide businesses with the choice to leverage the ‘right’ technologies, whether on-premises or in the cloud, truly putting the customer in control, and organizations that build on a proven set of service-oriented patterns and practices set themselves up for greater success in a Software + Services world.

This book is the result of mining and collating proven practices from the field. The authors have done an excellent job of explaining the architectural designs and goals behind SOA as well as real-world examples of SOA usage to build elegant IT solutions. It is my hope that this work plays a role in helping you realize loosely coupled, service-oriented solutions, on-premises and in the cloud, using Microsoft platforms and tools.

—S. Somasegar

Senior Vice President, Developer Division, Microsoft

Foreword by David Chappell

What is SOA? In the dozen or so years that the term has been around, service-oriented architecture has meant lots of different things. Some vendors saw it as a way to sell whatever it was they were offering, and so they jumped on the SOA bandwagon with both feet. Other people interpreted SOA in business terms, viewing it as a way to structure how different parts of an organization interact.

Yet from all of this confusion, one clear fact has emerged: The technology of service-orientation has real value. Whether or not it helps vendors sell products or managers organize their business, taking a service-oriented approach to development can make life better for the people who build and maintain applications.

The core reason for this is simple. Since applications rarely live out their lives in isolation, why not design those apps from the start with future connections in mind? Creating software that can expose and consume services marks the end of application silos, and it's fundamental to the value of service-orientation.

Doing this well requires two things. The first is a grasp of how to apply services effectively. Over time, our industry has evolved a set of patterns to help us create and connect applications in a service-oriented style. As with patterns in any other area, those for service-orientation attempt to codify best practices, helping all of us learn from what's gone before rather than reinvent these wheels on our own.

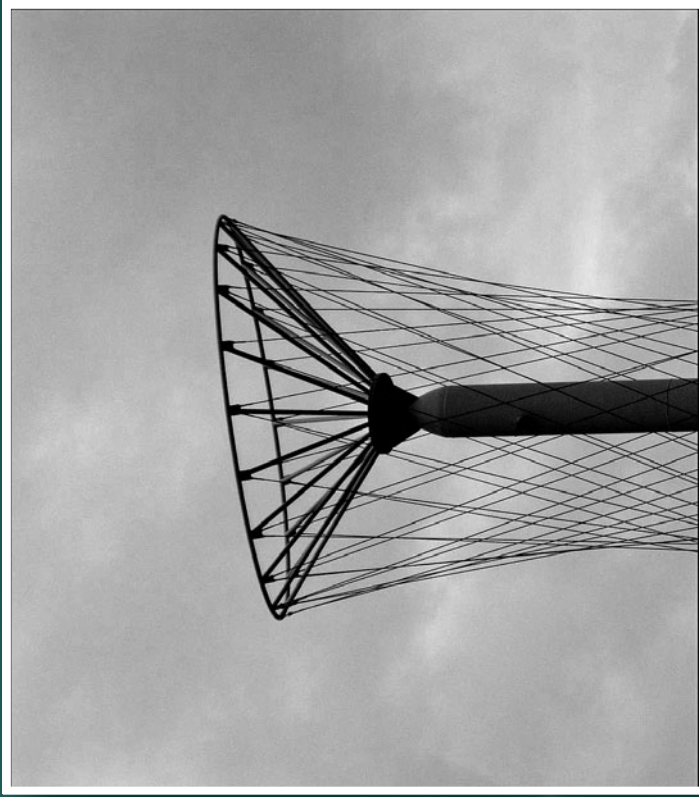
The second thing that's needed is a solid understanding of the technologies used to create service-oriented software. If you're working in the .NET world, there are quite a few things to get your mind around. Windows Communication Foundation provides a unified approach to creating and consuming services, for example, while Windows Workflow Foundation offers a generalized approach to orchestrating them. BizTalk Server takes an explicitly integration-oriented approach to the problem, while the Windows Azure platform brings services to the cloud. And even though the .NET Framework is common to all of these technologies, using them effectively isn't so easy. Each brings its own complexity to the party, and each can be combined with the others in various ways.

Explaining the intersection of these two worlds—service-orientation and .NET technologies—is exactly what this book does. Its team of specialist authors provides a concrete, usable guide to this combination, ranging from the fundamentals of service-orientation to the more rarified air of .NET services in the cloud and beyond. If you're creating service-oriented software on the Microsoft platform—that is, if you're a serious .NET developer—mastering these ideas is a must.

—David Chappell

Chappell & Associates (San Francisco, CA, USA)

Chapter 8



Cloud Services with Windows Azure

- 8.1 Cloud Computing 101
- 8.2 Windows Azure Platform Overview
- 8.3 Windows Azure Roles
- 8.4 Hello World in Windows Azure
- 8.5 A Web Service in Windows Azure
- 8.6 A REST Service in Windows Azure
- 8.7 Windows Azure Storage

Microsoft's Software-plus-Services strategy represents a view of the world where the growing feature-set of devices and the increasing ubiquity of the Web are combined to deliver more compelling solutions. Software-plus-Services represents an evolutionary step that is based on existing best practices in IT and extends the application potential of core service-orientation design principles.

Microsoft's efforts to embrace the Software-plus-Services vision are framed by three core goals:

- user experiences should span beyond a single device
- solution architectures should be able to intelligently leverage and integrate on-premise IT assets with cloud assets
- tightly coupled systems should give way to federations of cooperating systems and loosely coupled compositions

The Windows Azure platform represents one of the major components of the Software-plus-Services strategy, as Microsoft's cloud computing operating environment, designed from the outset to holistically manage pools of computation, storage and networking; all encapsulated by one or more services.

8.1 Cloud Computing 101

Just like service-oriented computing, cloud computing is a term that represents many diverse perspectives and technologies. In this book, our focus is on cloud computing in relation to SOA and Windows Azure.

Cloud computing enables the delivery of scalable and available capabilities by leveraging dynamic and on-demand infrastructure. By leveraging these modern service technology advances and various pervasive Internet technologies, the “cloud” represents an abstraction of services and resources, such that the underlying complexities of the technical implementations are encapsulated and transparent from users and consumer programs interacting with the cloud.

At the most fundamental level, cloud computing impacts two aspects of how people interact with technologies today:

- how services are consumed
- how services are delivered

Although cloud computing was originally, and still often is, associated with Web-based applications that can be accessed by end-users via various devices, it is also very much about applications and services themselves being consumers of cloud-based services. This fundamental change is a result of the transformation brought about by the adoption of SOA and Web-based industry standards, allowing for service-oriented and Web-based resources to become universally accessible on the Internet as on-demand services.

One example has been an approach whereby programmatic access to popular functions on Web properties is provided by simplifying efforts at integrating public-facing services and resource-based interactions, often via RESTful interfaces. This was also termed “Web-oriented architecture” or “WOA,” and was considered a subset of SOA. Architectural views such as this assisted in establishing the Web-as-a-platform concept, and helped shed light on the increasing inter-connected potential of the Web as a massive collection (or cloud) of ready-to-use and always-available capabilities.

This view can fundamentally change the way services are designed and constructed, as we reuse not only someone else’s code and data, but also their infrastructure resources, and leverage them as part of our own service implementations. We do not need to understand the inner workings and technical details of these services; Service Abstraction (696), as a principle, is applied to its fullest extent by hiding implementation details behind clouds.

With regards to service delivery, we are focused on the actual design, development, and implementation of cloud-based services. Let’s begin by establishing high-level characteristics that a cloud computing environment can include:

- generally accessible
- always available and highly reliable
- elastic and scalable
- abstract and modular resources

SOA PRINCIPLES & PATTERNS

There are several SOA design patterns that are closely related to common cloud computing implementations, such as Decoupled Contract [735], Redundant Implementation [766], State Repository [785], and Stateful Services [786]. In this and subsequent chapters, these and other patterns will be explored as they apply specifically to the Windows Azure cloud platform.

- service-oriented
- self-service management and simplified provisioning

Fundamental topics regarding service delivery pertain to the cloud deployment model used to provide the hosting environment and the service delivery model that represents the functional nature of a given cloud-based service. The next two sections explore these two types of models.

Cloud Deployment Models

There are three primary cloud deployment models. Each can exhibit the previously listed characteristics; their differences lie primarily in the scope and access of published cloud services, as they are made available to service consumers.

Let's briefly discuss these deployment models individually.

Public Cloud

Also known as external cloud or multi-tenant cloud, this model essentially represents a cloud environment that is openly accessible. It generally provides an IT infrastructure in a third-party physical data center that can be utilized to deliver services without having to be concerned with the underlying technical complexities.

Essential characteristics of a public cloud typically include:

- homogeneous infrastructure
- common policies
- shared resources and multi-tenant
- leased or rented infrastructure; operational expenditure cost model
- economies of scale and elastic scalability

Note that public clouds can host individual services or collections of services, allow for the deployment of service compositions, and even entire service inventories.

Private Cloud

Also referred to as internal cloud or on-premise cloud, a private cloud intentionally limits access to its resources to service consumers that belong to the same organization that owns the cloud. In other words, the infrastructure that is managed and operated for one

organization only, primarily to maintain a consistent level of control over security, privacy, and governance.

Essential characteristics of a private cloud typically include:

- heterogeneous infrastructure
- customized and tailored policies
- dedicated resources
- in-house infrastructure (capital expenditure cost model)
- end-to-end control

Community Cloud

This deployment model typically refers to special-purpose cloud computing environments shared and managed by a number of related organizations participating in a common domain or vertical market.

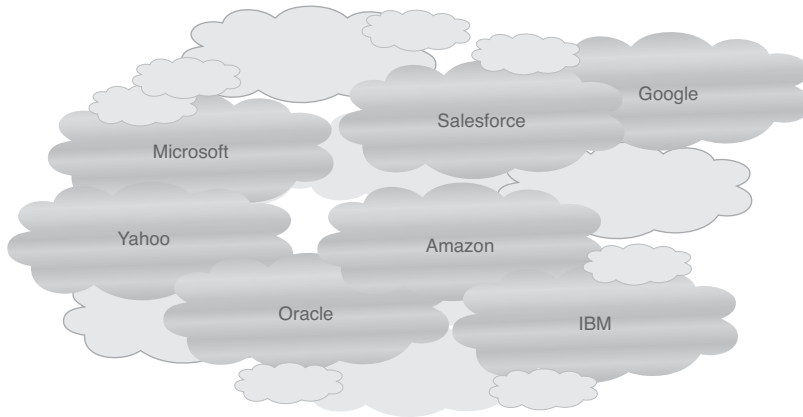
Other Deployment Models

There are variations of the previously discussed deployment models that are also worth noting. The *hybrid cloud*, for example, refers to a model comprised of both private and public cloud environments. The *dedicated cloud* (also known as the hosted cloud or virtual private cloud) represents cloud computing environments hosted and managed off-premise or in public cloud environments, but dedicated resources are provisioned solely for an organization's private use.

The Intercloud (Cloud of Clouds)

The intercloud is not as much a deployment model as it is a concept based on the aggregation of deployed clouds (Figure 8.1). Just like the Internet, which is a network of networks; intercloud refers to an inter-connected global cloud of clouds. Also like the World Wide Web, intercloud represents a massive collection of services that organizations can explore and consume.

From a services consumption perspective, we can look at the intercloud as an on-demand SOA environment where useful services managed by other organizations can be leveraged and composed. In other words, services that are outside of an organization's own boundaries and operated and managed by others can become a part of the aggregate portfolio of services of those same organizations.

**Figure 8.1**

Examples of how vendors establish a commercial intercloud.

Deployment Models and Windows Azure

Windows Azure exists in a public cloud. Windows Azure itself is not made available as a packaged software product for organizations to deploy into their own IT enterprises. However, Windows Azure-related features and extensions exist in Microsoft's on-premise software products, and are collectively part of Microsoft's private cloud strategy. It is important to understand that even though the software infrastructure that runs Microsoft's public cloud and private clouds are different, layers that matter to end-user organizations, such as management, security, integration, data, and application are increasingly consistent across private and public cloud environments.

Service Delivery Models

Many different types of services can be delivered in the various cloud deployment environments. Essentially, any IT resource or function can eventually be made available as a service. Although cloud-based ecosystems allow for a wide range of service delivery models, three have become most prominent:

Infrastructure-as-a-Service (IaaS)

This service delivery model represents a modern form of utility computing and outsourced managed hosting. IaaS environments manage and provision fundamental computing resources (networking, storage, virtualized servers, etc.). This allows consumers to deploy and manage assets on leased or rented server instances, while the service providers own and govern the underlying infrastructure.

Platform-as-a-Service (PaaS)

The PaaS model refers to an environment that provisions application platform resources to enable direct deployment of application-level assets (code, data, configurations, policies, etc.). This type of service generally operates at a higher abstraction level so that users manage and control the assets they deploy into these environments. With this arrangement, service providers maintain and govern the application environments, server instances, as well as the underlying infrastructure.

Software-as-a-Service (SaaS)

Hosted software applications or multi-tenant application services that end-users consume directly correspond to the SaaS delivery model. Consumers typically only have control over how they use the cloud-based service, while service providers maintain and govern the software, data, and underlying infrastructure.

Other Delivery Models

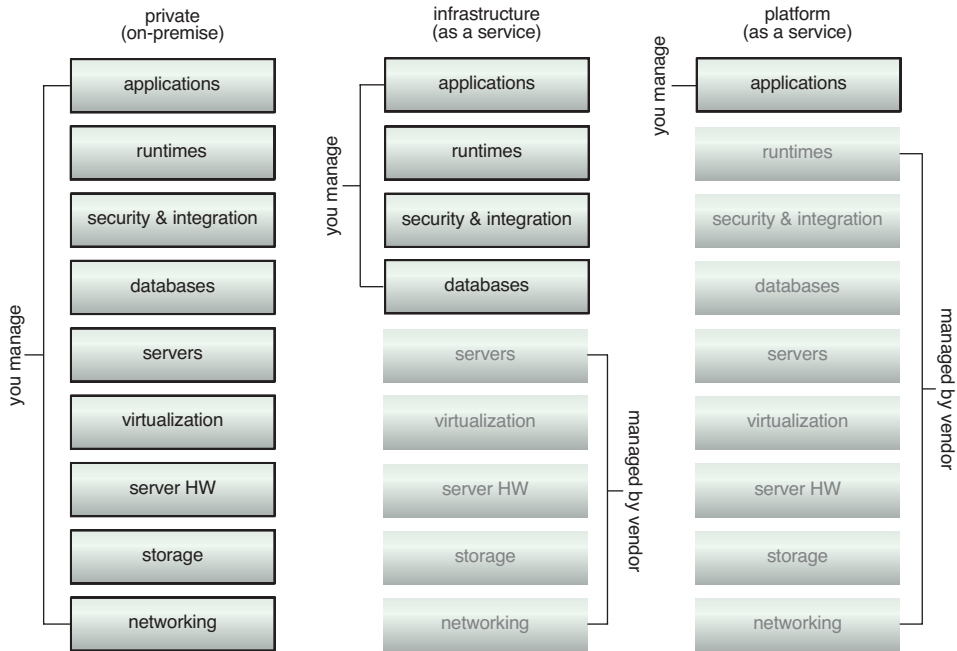
Cloud computing is not limited to the aforementioned delivery models. Security, governance, business process management, integration, complex event processing, information and data repository processing, collaborative processes—all can be exposed as services and consumed and utilized to create other services.

NOTE

Cloud deployment models and service delivery models are covered in more detail in the upcoming book *SOA & Cloud Computing* as part of the *Prentice Hall Service-Oriented Computing Series* from Thomas Erl. This book will also introduce several new design patterns related to cloud-based service, composition, and platform design.

IaaS vs. PaaS

In the context of SOA and developing cloud-based services with Windows Azure, we will focus primarily on IaaS and PaaS delivery models in this chapter. Figure 8.2 illustrates a helpful comparison that contrasts some primary differences. Basically, IaaS represents a separate environment to host the same assets that were traditionally hosted on-premise, whereas PaaS represents environments that can be leveraged to build and host next-generation service-oriented solutions.

**Figure 8.2**

Common differentiations between delivery models.

We interact with PaaS at a higher abstraction level than with IaaS. This means we manage less of the infrastructure and assume simplified administration responsibilities. But at the same time, we have less control over this type of environment.

IaaS provides a similar infrastructure to traditional on-premise environments, but we may need to assume the responsibility to re-architect an application in order to effectively leverage platform service clouds. In the end, PaaS will generally achieve a higher level of scalability and reliability for hosted services.

IN PLAIN ENGLISH

An on-premise infrastructure is like having your own car. You have complete control over when and where you want to drive it, but you are also responsible for its operation and maintenance. IaaS is like using a car rental service. You still have control over when and where you want to go, but you don't need to be concerned with the vehicle's maintenance. PaaS is more comparable to public transportation. It is easier to use as you don't need to know how to operate it and it costs less. However, you don't have control over its operation, schedule, or routes.

SUMMARY OF KEY POINTS

- Cloud computing enables the delivery of scalable and available capabilities by leveraging dynamic and on-demand infrastructure.
- There are three common types of cloud deployment models: public cloud, private cloud, and community cloud.
- There are three common types of service delivery models: IaaS, PaaS, and SaaS.

8.2 Windows Azure Platform Overview

The Windows Azure platform is an Internet-scale cloud computing services platform hosted in Microsoft data centers. Windows tools provide functionality to build solutions that include a cloud services operating system and a set of developer services. The key parts of the Windows Azure platform are:

- Windows Azure (application container)
- Microsoft SQL Azure
- Windows Azure platform AppFabric

The Windows Azure platform is part of the Microsoft cloud, which consists of multiple categories of services:

- *cloud-based applications* – These are services that are always available and highly scalable. They run in the Microsoft cloud that consumers can directly utilize. Examples include Bing, Windows Live Hotmail, Office Live, etc.
- *software services* – These services are hosted instances of Microsoft's enterprise server products that consumers can use directly. Examples include Exchange Online, SharePoint Online, Office Communications Online, etc.

SOA PRINCIPLES & PATTERNS

The infrastructure and service architectures that underlie many of these native services (as well as cloud-based services in general) are based on direct combined application of Stateful Services [786] and Redundant Implementation [766]. This is made possible by leveraging several of the built-in extensions and mechanisms provided by the Windows Azure platform (as explained in this chapter and Chapter 16).

- *platform services* – This is where the Windows Azure platform itself is positioned. It serves as an application platform public cloud that developers can use to deploy next-generation, Internet-scale, and always available solutions.
- *infrastructure services* – There is a limited set of elements of the Windows Azure platform that can support cloud-based infrastructure resources.

Figure 8.3 illustrates the service categories related to the Windows Azure platform. Given that Windows Azure is itself a platform, let's explore it as an implementation of the PaaS delivery model.

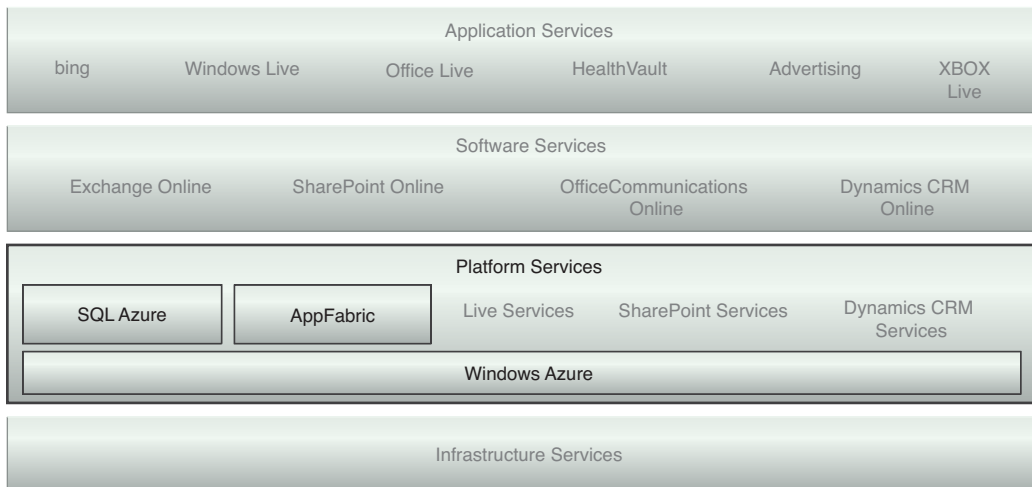


Figure 8.3

A high-level representation of categories of services available in the Windows Azure cloud.

The Windows Azure platform was built from the ground up using Microsoft technologies, such as the Windows Server Hyper-V-based system virtualization layer. However, the Windows Azure platform is not intended to be just another off-premise Windows Server hosting environment. It has a cloud fabric layer, called the *Windows Azure Fabric Controller*, built on top of its underlying infrastructure.

The Windows Azure Fabric Controller pools an array of virtualized Windows Server instances into a logical entity and automatically manages the following:

- resources
- load balancing
- fault-tolerance

- geo-replication
- application lifecycle

These are managed without requiring the hosted applications to explicitly deal with the details. The fabric layer provides a parallel management system that abstracts the complexities in the infrastructure and presents a cloud environment that is inherently elastic. As a form of PaaS, it also supports the access points for user and application interactions with the Windows Azure platform.

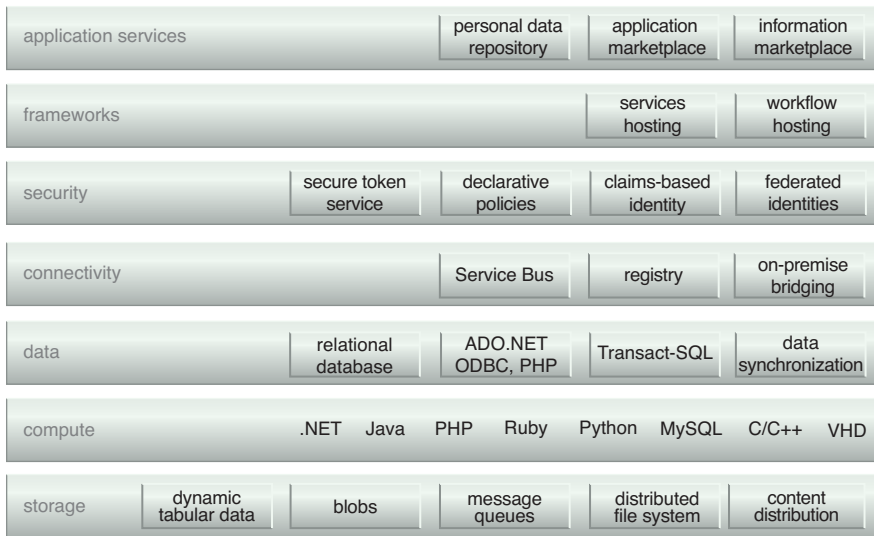


Figure 8.4

An overview of common Windows Azure platform capabilities.

The Windows Azure platform essentially provides a set of cloud-based services that are symmetric with existing mainstream on-site enterprise application platforms (Figure 8.4).

For example:

- *storage services* – a scalable distributed data storage system that supports many types of storage models, including hash map or table-like structured data, large binary files, asynchronous messaging queues, traditional file systems, and content distribution networks
- *compute services* – application containers that support existing mainstream development technologies and frameworks, including .NET, Java, PHP, Python, Ruby on Rails, and native code.

- *data services* – highly reliable and scalable relational database services that also support integration and data synchronization capabilities with existing on-premise relational databases
- *connectivity services* – these are provided via a cloud-based service bus that can be used as a message intermediary to broker connections with other cloud-based services and services behind firewalls within on-premise enterprise environments
- *security services* – policy-driven access control services that are federation-aware and can seamlessly integrate with existing on-premise identity management systems
- *framework services* – components and tools that support specific aspects and requirements of solution frameworks
- *application services* – higher-level services that can be used to support application development, such as application and data marketplaces

All of these capabilities can be utilized individually or in combination.

Windows Azure (Application Container)

Windows Azure serves as the development, service hosting, and service management environment. It provides the application container into which code and logic, such as Visual Studio projects, can be deployed. The application environment is similar to existing Windows Server environments. In fact, most .NET projects can be deployed directly without significant changes.

A Windows Azure instance represents a unit of deployment, and is mapped to specific virtual machines with a range of variable sizes. Physical provisioning of the Windows Azure instances is handled by the cloud fabric. We are required only to specify, by policy, how many instances we want the cloud fabric to deploy for a given service.

We have the ability to manually start and shut down instances, and grow or shrink the deployment pool; however, the cloud fabric also provides automated management of the health and lifecycles of instances. For example, in the event of an instance failure, the cloud fabric would automatically shut down the instance and attempt to bring it back up on another node.

Windows Azure also provides a set of storage services that consumers can use to store and manage persistent and transient data. Storage services support geo-location and offer high durability of data by triple-replicating everything within a cluster and across

data centers. Furthermore, they can manage scalability requirements by automatically partitioning and load balancing services across servers.

Also supported by Windows Azure is a VHD-based deployment model as an option to enable some IaaS requirements. This is primarily geared for services that require closer integration with the Windows Server OS. This option provides more control over the service hosting environment and can better support legacy applications.

SQL Azure

SQL Azure is a cloud-based relational database service built on SQL Server technologies that exposes a fault-tolerant, scalable, and multi-tenant database service. SQL Azure does not exist as hosted instances of SQL Server. It also uses a cloud fabric layer to abstract and encapsulate the underlying technologies required for provisioning, server administration, patching, health monitoring, and lifecycle management. We are only required to deal with logical administration tasks, such as schema creation and maintenance, query optimization, and security management.

A SQL Azure database instance is actually implemented as three replicas on top of a shared SQL Server infrastructure managed by the cloud fabric. This cloud fabric delivers high availability, reliability, and scalability with automated and transparent replication

SOA PRINCIPLES & PATTERNS

Services deployed within Windows Azure containers and made available via Windows Azure instances establish service architectures that, on the surface, resemble typical Web service or REST service implementations. However, the nature of the back-end processing is highly extensible and scalable and can be further subject to various forms of Service Refactoring [783] over time to accommodate changing usage requirements. This highlights the need for Windows Azure hosted services to maintain the freedom to be independently governed and evolved. This, in turn, places a greater emphasis on the balanced design of the service contract and its proper separation as part of the overall service architecture.

Specifically, it elevates the importance of the Standardized Service Contract (693), Service Loose Coupling (695), and Service Abstraction (696) principles that, through collective application, shape and position service contracts to maximize abstraction and cross-service standardization, while minimizing negative forms of consumer and implementation coupling. Decoupled Contract [735] forms an expected foundation for Windows Azure-hosted service contracts, and there will generally be the need for more specialized contract-centric patterns, such as Validation Abstraction [792], Canonical Schema [718], and Schema Centralization [769].

and failover. It further supports load-balancing of consumer requests and the synchronization of concurrent, incremental changes across the replicas. The cloud fabric also handles concurrency conflict resolutions when performing bi-directional data synchronization between replicas by using built-in policies (such as *last-writer-wins*) or custom policies.

Because SQL Azure is built on SQL Server, it provides a familiar relational data model and is highly symmetric to on-premise SQL Server implementations. It supports most features available in the regular SQL Server database engine and can also be used with tools like SQL Server 2008 Management Studio, SQLCMD, and BCP, and SQL Server Integration Services for data migration.

SOA PRINCIPLES & PATTERNS

In addition to reliability and scalability improvements, SQL Azure's replication mechanism can be used to apply Service Data Replication [773] in support of the Service Autonomy (699) principle. This is significant, as individual service autonomy within cloud environments can often fluctuate due to the heavy emphasis on shared resources across pools of cloud-based services.

Windows Azure Platform AppFabric

In Chapter 7, as part of our coverage of .NET Enterprise Services, we introduced Windows Server AppFabric. This represents the version of AppFabric that is local to the Windows Server environment. *Windows Azure platform AppFabric* (with the word “platform” intentionally not capitalized), is the cloud-based version of AppFabric that runs on Windows Azure.

Windows Azure platform AppFabric helps connect services within or across clouds and enterprises. It provides a Service Bus for connectivity across networks and organizational boundaries, and an Access Control service for federated authorization as a service.

The Service Bus acts as a centralized message broker in the cloud to relay messages between services and service consumers. It has the ability to connect to on-premise services through firewalls, NATs, and over any network topology.

Its features include:

- connectivity using standard protocols and standard WCF bindings
- multiple communication models (such as publish-and-subscribe, one-way messaging, unicast and multicast datagram distribution, full-duplex bi-directional connection-oriented sessions, peer-to-peer sessions, and end-to-end NAT traversal)

- service endpoints that are published and discovered via Internet-accessible URLs
- global hierarchical namespaces that are DNS and transport-independent
- built-in intrusion detection and protection against denial-of-service attacks

Access Control acts as a centralized cloud-based security gateway that regulates access to cloud-based services and Service Bus communications, while integrating with standards-based identity providers (including enterprise directories such as Active Directory and online identity systems like Windows Live ID). Access Control and other Windows Azure-related security topics are covered in Chapter 17.

Unlike Windows Azure and SQL Azure, which are based on Windows Server and SQL Server, Access Control Service is not based on an existing server product. It uses technology included in Windows Identity Foundation and is considered a purely cloud-based service built specifically for the Windows Azure platform environment.

SOA PRINCIPLES & PATTERNS

The Windows Azure Service Bus complies to the familiar Enterprise Service Bus [741] compound pattern, and focuses on realizing this pattern across network, security, and organizational domains.

Service Bus also provides a service registry to provide registration and discovery of service metadata, which allows for the application of Metadata Centralization [754] and emphasizes the need to apply the Service Discoverability (702) principle.

SUMMARY OF KEY POINTS

- The Windows Azure platform is primarily a PaaS deployed in a public cloud managed by Microsoft.
- Windows Azure platform provides a distinct set of capabilities suitable for building scalable and reliable cloud-based services.
- The overall Windows Azure platform further encompasses SQL Azure and Windows Azure platform AppFabric.

8.3 Windows Azure Roles

A cloud service in Windows Azure will typically have multiple concurrent instances. Each instance may be running all or a part of the service's codebase. As a developer, you control the number and type of roles that you want running your service.

Web Roles and Worker Roles

Windows Azure roles are comparable to standard Visual Studio projects, where each instance represents a separate project. These roles represent different types of applications that are natively supported by Windows Azure. There are two types of roles that you can use to host services with Windows Azure:

- Web roles
- worker roles

Web roles provide support for HTTP and HTTPS through public endpoints and are hosted in IIS. They are most comparable to regular ASP.NET projects, except for differences in their configuration files and the assemblies they reference.

Worker roles can also expose external, publicly facing TCP/IP endpoints on ports other than 80 (HTTP) and 443 (HTTPS); however, worker roles do not run in IIS. Worker roles are applications comparable to Windows services and are suitable for background processing.

Virtual Machines

Underneath the Windows Azure platform, in an area that you and your service logic have no control over, each role is given its own virtual machine or VM. Each VM is created when you deploy your service or service-oriented solution to the cloud. All of these VMs are managed by a modified hypervisor and hosted in one of Microsoft's global data centers.

Each VM can vary in size, which pertains to the number of CPU cores and memory. This is something that you control. So far, four pre-defined VM sizes are provided:

- small – 1.7ghz single core, 2GB memory
- medium – 2x 1.7ghz cores, 4GB memory
- large – 4x 1.7ghz cores, 8GB memory
- extra large – 8x 1.7ghz cores, 16GB memory

Notice how each subsequent VM on this list is twice as big as the previous one. This simplifies VM allocation, creation, and management by the hypervisor.

Windows Azure abstracts away the management and maintenance tasks that come along with traditional on-premise service implementations. When you deploy your service into Windows Azure and the service's roles are spun up, copies of those roles are

replicated automatically to handle failover (for example, if a VM were to crash because of hard drive failure). When a failure occurs, Windows Azure automatically replaces that “unreliable” role with one of the “shadow” roles that it originally created for your service.

This type of failover is nothing new. On-premise service implementations have been leveraging it for some time using clustering and disaster recovery solutions. However, a common problem with these failover mechanisms is that they are often server-focused. This means that the entire server is failed over, not just a given service or service composition.

When you have multiple services hosted on a Web server that crashes, each hosted service experiences downtime between the current server crashing and the time it takes to bring up the backup server. Although this may not affect larger organizations with sophisticated infrastructure too much, it can impact smaller IT enterprises that may not have the capital to invest in setting up the proper type of failover infrastructure.

Also, suppose you discover in hindsight after performing the failover that it was some background worker process that caused the crash. This probably means that unless you can address it quick enough, your failover server is under the same threat of crashing.

Windows Azure addresses this issue by focusing on application and hosting roles. Each service or solution can have a Web frontend that runs in a Web role. Even though each role has its own “active” virtual machine (assuming we are working with single instances), Windows Azure creates copies of each role that are physically located on one or more servers. These servers may or may not be running in the same data center. These shadow VMs remain idle until they are needed.

Should the background process code crash the worker role and subsequently put the underlying virtual machine out of commission, Windows Azure detects this and automatically brings in one of the shadow worker roles. The faulty role is essentially discarded. If the worker role breaks again, then Windows Azure replaces it once more. All of this is happening without any downtime to the solution’s Web role front end, or to any other services that may be running in the cloud.

Input Endpoints

Web roles used to be the only roles that could receive Internet traffic, but now worker roles can listen to any port specified in the service definition file. Internet traffic is received through the use of *input endpoints*. Input endpoints and their listening ports are declared in the service definition (*.csdef) file.

Keep in mind that when you specify the port for your worker role to listen on, Windows Azure isn't actually going to assign that port to the worker. In reality, the load balancer will open two ports—one for the Internet and the other for your worker role. Suppose you wanted to create an FTP worker role and in your service definition file you specify port 21. This tells the fabric load balancer to open port 21 on the Internet side, open pseudo-random port 33476 on the LAN side, and begin routing FTP traffic to the FTP worker role.

In order to find out which port to initialize for the randomly assigned internal port, use the `RoleEnvironment.CurrentRoleInstance.InstanceEndpoints["FtpIn"].IPEndpoint` object.

Inter-Role Communication

Inter-Role Communication (IRC) allows multiple roles to talk to each other by exposing internal endpoints. With an internal endpoint, you specify a name instead of a port number. The Windows Azure application fabric will assign a port for you automatically and will also manage the name-to-port mapping.

Here is an example of how you would specify an internal endpoint for IRC:

```
<ServiceDefinition xmlns=
  "http://schemas.microsoft.com/ServiceHosting/2008/10/
  ServiceDefinition" name="HelloWorld">
  <WorkerRole name="WorkerRole1">
    <Endpoints>
      <InternalEndpoint name="NotifyWorker" protocol="tcp" />
    </Endpoints>
  </WorkerRole>
</ServiceDefinition>
```

Example 8.1

In this example, `NotifyWorker` is the name of the internal endpoint of a worker role named `WorkerRole1`. Next, you need to define the internal endpoint, as follows:

```
RoleInstanceEndpoint internalEndPoint =
RoleEnvironment.CurrentRoleInstance.
  InstanceEndpoints["NotificationService"];
this.serviceHost.AddServiceEndpoint(
  typeof(INameOfYourContract),
  binding,
```

```
String.Format("net.tcp://{0}/NotifyWorker",  
    internalEndPoint.IPEndpoint));  
WorkerRole.factory = new ChannelFactory<IClientNotification>(binding);
```

Example 8.2

You only need to specify the IP endpoint of the other worker role instances in order to communicate with them. For example, you could get a list of these endpoints with the following routine:

```
var current = RoleEnvironment.CurrentRoleInstance;  
var endPoints = current.Role.Instances  
    .Where(instance => instance != current)  
    .Select(instance => instance.InstanceEndpoints["NotifyWorker"]);
```

Example 8.3

IRC only works for roles in a single application deployment. Therefore, if you have multiple applications deployed and would like to enable some type of cross-application role communication, IRC won't work. You will need to use queues instead.

SUMMARY OF KEY POINTS

- Windows Azure roles represent different types of supported applications or services.
 - There are two types of roles: Web roles and worker roles.
 - Each role is assigned its own VM.
-

8.4 Hello World in Windows Azure

The following section demonstrates the creation of a simple “Hello World” service in a Windows Azure hosted application.

NOTE

If you are carrying out the upcoming steps with Visual Studio 2008, you will need to be in an elevated mode (such as Administrator). A convenient way of determining whether the mode setting is correct is to press the F5 key in order to enter debug mode. If you receive an error stating *“the development fabric must be run elevated,”* then you will need to restart Visual Studio as an administrator.

Also, ensure the following on your SQL Express setup:

- SQL Server Express Edition 2008 must be running under the ‘.\SQL-EXPRESS’ instance
- your Windows account must have a login in .\SQLEXPRESS
- your login account is a member of the sysadmin role

If SQL Express isn’t configured properly, you will get a permissions error.

1. Create a Cloud Service Project

First you need to open the New Project window to create a new cloud service project using VB.NET or C# (Figure 8.5).

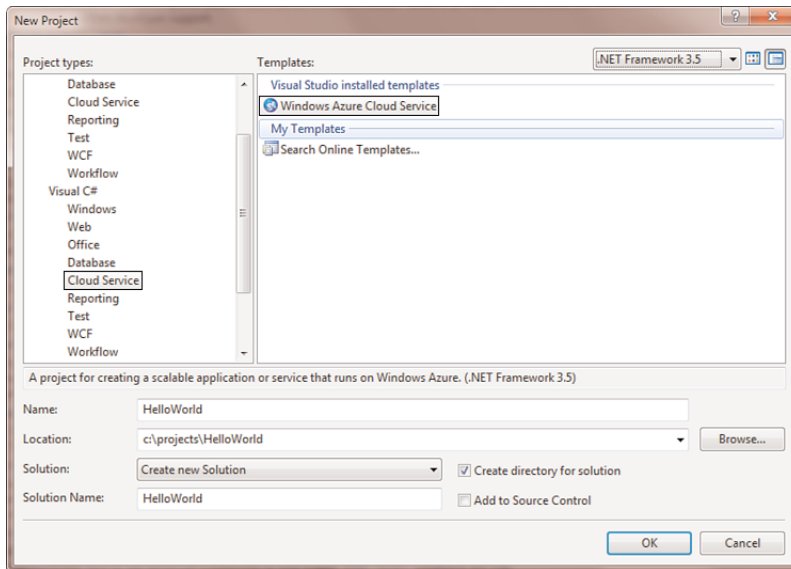


Figure 8.5

The New Project window.

2. Choose an ASP.NET Web Role

After you click OK on the New Project window, the New Cloud Service Project wizard will start. You will then see a window (Figure 8.6) that will allow you to choose the type of role that you would like as part of your service deployment.

For the Hello World project, you will only need the ASP.NET Web Role type. Once you select this role, you can choose the role name.

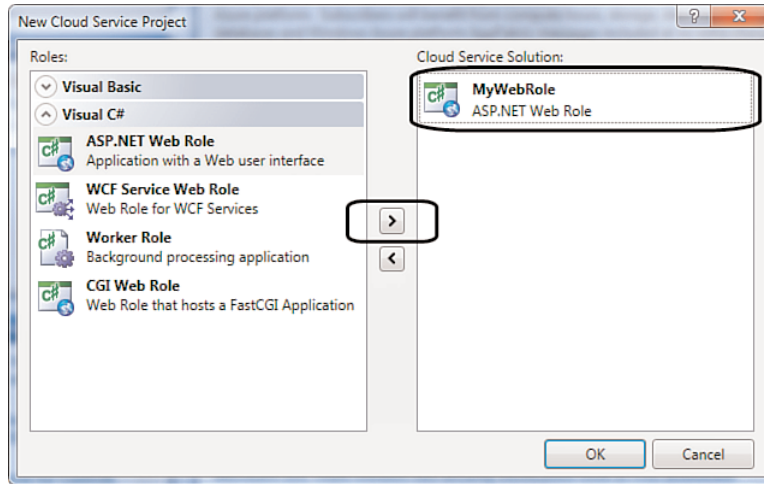


Figure 8.6
The New Cloud Service Project window.

3. Create the Solution

After clicking OK, the wizard will generate the solution, which you can then view using the Solution Explorer window (Figure 8.7).

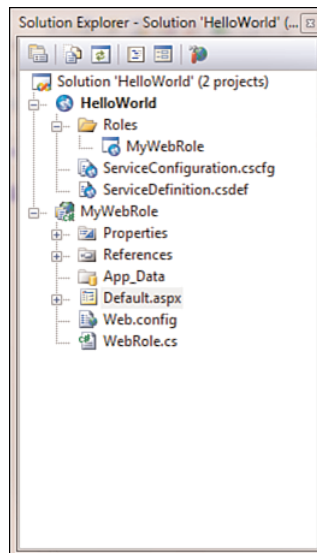


Figure 8.7
The HelloWorld solution structure displayed in the Solution Explorer window.

4. Instantiate the Service

Now you can open the Default.aspx file using the Solution Explorer window, put “Hello, Cloud!” in the Body element and press F5 to run. You should see something like what is shown in Figure 8.8.

This example was executed locally on IIS. If we were to deploy this service into the Windows Azure cloud, it would still be running in IIS because it is hosted in a Web role.

SOA PRINCIPLES & PATTERNS

Mainstream SOA design patterns and service-orientation principles can be applied to Windows Azure-hosted services very similarly to how they are applied to internal enterprise-hosted services. Furthermore, Windows Azure-hosted services support different service implementation mediums (such as Web services and REST services) and allow for the same service to be accessed via multiple protocols. This supports the creative application of specialized patterns, such as Concurrent Contracts [726] and Dual Protocols [739].

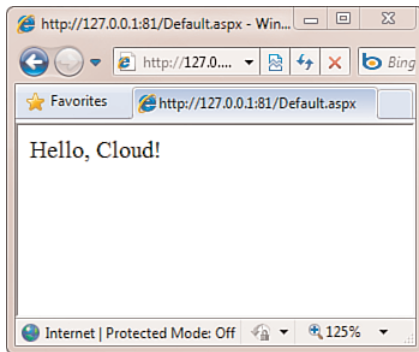


Figure 8.8

The HelloWorld service in action.

SUMMARY OF KEY POINTS

- The development environment for Windows Azure is fully integrated into Visual Studio, which provides a simulated runtime for Windows Azure for local desktop-based development and unit testing.
- Creating and deploying cloud-based services with Windows Azure is simplified using available wizards and development UIs.

8.5 A Web Service in Windows Azure

In this section example, we take a closer look at a Web service that is deployed to Windows Azure in order to better understand the code-level impacts of moving a service to a cloud.

Let's assume we moved a service contract interface definition and a data contract into a custom C# project. We choose `ServiceClient` to test our service and `ServiceDemo` contains the Windows Azure application configuration and definition files.

We further opt to host this project in a Web role, which means that there is a little bit of bootstrapping that needs to be done. The `WebRole` class inherits from the `RoleEntryPoint` class, which contains methods that are used by Windows Azure to start or stop the role. You can optionally override those methods to manage the initialization or shut-down process of your role. Worker roles must extend `RoleEntryPoint`, but it is optional for Web roles. The Visual Studio tools will automatically extend this class for you, as you can see from the `WebRole.cs` code:

```
using System.Linq;
using Microsoft.WindowsAzure.Diagnostics;
using Microsoft.WindowsAzure.ServiceRuntime;
using System.Diagnostics;
namespace ServiceDemo_WebRole
{
    public class WebRole : RoleEntryPoint
    {
        public override bool OnStart()
        {
            DiagnosticMonitor.Start("DiagnosticsConnectionString");
            RoleEnvironment.Changing += RoleEnvironmentChanging;
            Trace.TraceInformation("WebRole starting...");
            return base.OnStart();
        }
        private void RoleEnvironmentChanging(object sender,
            RoleEnvironmentChangingEventArgs e)
        {
            if (e.Changes.Any(change => change is
                RoleEnvironmentConfigurationSettingChange))
            {
                e.Cancel = true;
            }
        }
    }
}
```

Example 8.4

Our cloud service project includes two configuration files: `ServiceDefinition.csdef` and `ServiceConfiguration.cscfg`. These files are packaged together with the cloud service when it is deployed to Windows Azure.

The `ServiceDefinition.csdef` file contains the metadata needed by the Windows Azure environment to understand the requirements of the service, including the roles it contains. It also establishes configuration settings that will be applied to all specified service roles:

```
<ServiceDefinition name="ServiceDemo" xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceDefinition">
  <WebRole name="ServiceDemo_WebRole">
    <InputEndpoints>
      <InputEndpoint name="HttpIn" protocol="http" port="80" />
    </InputEndpoints>
    <ConfigurationSettings>
      <Setting name="DiagnosticsConnectionString" />
    </ConfigurationSettings>
  </WebRole>
</ServiceDefinition>
```

Example 8.5

The `ServiceConfiguration.cscfg` file sets values for the configuration settings defined in the service definition file and specifies the number of instances to run for each role. Here is the `ServiceConfiguration.cscfg` for the `ServiceDemo` service project:

```
<ServiceConfiguration serviceName="ServiceDemo"
  xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceConfiguration">
  <Role name="ServiceDemo_WebRole">
    <Instances count="2" />
    <ConfigurationSettings>
      <Setting name="DiagnosticsConnectionString"
        value="UseDevelopmentStorage=true" />
    </ConfigurationSettings>
  </Role>
</ServiceConfiguration>
```

Example 8.6

The `Instances` element tells the Windows Azure runtime fabric how many instances to spin up for the `ServiceDemo_WebRole` role. By default, Visual Studio tools set this to "1", but this is generally not a good idea. If you only have one role running and it crashes, it could take a while before Windows Azure spins up another one. However, if you had multiple roles and one goes down, the application wouldn't experience a work stop while a new instance is being generated. This is why it is a good practice to have at least two role instances per role.

In the `ConfigurationSettings` section, there is a statement worth singling out:

```
<Setting name="DiagnosticsConnectionString"
  value="UseDevelopmentStorage=true" />
```

Example 8.7

There is a set of logging and diagnostic APIs that you can use to instrument your code and provide better traceability. With these APIs, you can not only detect and troubleshoot problems, but you can also gain insight into the overall performance of an application.

This line of code passes in the configuration setting name that is equal to the connection string for the storage account that the Diagnostic Monitor needs to use to store the diagnostic data. By default, the setting name is `DiagnosticsConnectionString`, but you can name it whatever you like as long as the name matches up with the service definition and service configuration files.

In the `WebRole.cs`, you will see the following statement:

```
DiagnosticMonitor.Start("DiagnosticsConnectionString");
```

Example 8.8

This line of code starts up the Diagnostic Monitor when the role starts. By default, the connection string is set to use development storage, such as the SQL table that was created when the SDK was installed. Before you deploy the service to the Windows Azure cloud, you will need to update this setting with the storage account name and account key information.

For example:

```
<ConfigurationSettings>
  <Setting name="DiagnosticsConnectionString"
    value="DefaultEndpointsProtocol=https;AccountName=
      [ACCOUNT NAME];AccountKey=[ACCOUNT KEY]" />
</ConfigurationSettings>
```

Example 8.9

If we take a look at the Web role's Web.Config file, we'll also see that the project wizard automatically created the following:

```
<system.diagnostics>
  <trace>
    <listeners>
      <add type="Microsoft.WindowsAzure.Diagnostics.
        DiagnosticMonitorTraceListener,
        Microsoft.WindowsAzure.Diagnostics, Version=1.0.0.0,
        Culture=neutral, PublicKeyToken=31bf3856ad364e35"
        name="AzureDiagnostics">
        <filter type="" />
      </add>
    </listeners>
  </trace>
</system.diagnostics>
```

Example 8.10

This creates a tracing listener for the diagnostic monitor, which means that we continue to use the `System.Diagnostics.Trace` class for instrumentation. The diagnostic monitor will just hook into those calls and push them into storage.

The following examples show the `IOrderService` interface contract and the `Order` data contract, followed by the final output:

```
namespace Contract
{
  [ServiceContract]
  public interface IOrderService
  {
    [OperationContract]
    int CreateOrder(Order o);
    [OperationContract]
    void UpdateOrder(string id, Order o);
    [OperationContract]
  }
}
```

```

    Order GetOrderByOrderId(string id);
    [OperationContract]
    List<Order> GetOrdersByCustomer(string custName);
    [OperationContract]
    List<Order> GetOrders();
    [OperationContract]
    void DeleteOrder(string id);
}
}

```

Example 8.11

```

namespace Contract
{
    [DataContract(Namespace=
        "http://example.cloudapp.net/servicedemo/1.0")]
    public class Order
    {
        [DataMember]
        public int OrderId { get; set; }
        [DataMember]
        public string OrderItem { get; set; }
        [DataMember]
        public string CustomerName { get; set; }
    }
}

```

Example 8.12

```

namespace ServiceDemo_WebRole
{
    [ServiceBehavior(InstanceContextMode =
        InstanceContextMode.Single,
        AddressFilterMode =
        AddressFilterMode.Any)]
    public class OrderService : Contract.IOrderService
    {
        int id = 0;
        List<Order> Orders = new List<Order>();
        #region IOrderService Members
        int IOrderService.CreateOrder(Order o)
        {
            o.OrderId = ++id;
            Orders.Add(o);
            return o.OrderId;
        }
    }
}

```

```

    }
    void IOrderService.UpdateOrder(string id, Order o)
    {
        var first = Orders.First(order =>
            order.OrderId ==
            Convert.ToInt64(id));
        first = o;
    }
    List<Order> IOrderService.GetOrders()
    {
        return Orders;
    }
    void IOrderService.DeleteOrder(string orderId)
    {
        Orders.RemoveAll(order =>
            order.OrderId.Equals
            (Convert.ToInt64(orderId)));
    }
    Order IOrderService.GetOrderByOrderId(string orderId)
    {
        return Orders.First(o =>
            o.OrderId.Equals(Convert.ToInt64(orderId)));
    }
    public List<Order> GetOrdersByCustomer(string custName)
    {
        return (string.IsNullOrEmpty(custName)) ?
            Orders : Orders.FindAll(o =>
                o.CustomerName.Equals(custName));
    }
    #endregion
}
}

```

Example 8.13

Note that the `InstanceContextMode` setting is set to `single` because we want to use the same service object instance across the communication session established between the service and its consumer. In a real world scenario, you would choose a more robust solution like SQL Azure or Windows Azure table storage (covered later in this chapter). Let's briefly walk through the steps required to actually deploy the service to Windows Azure.

1. Create a Host Service and Storage Service

When you create a storage service, you have to create a globally unique storage account name, not to be confused with the overarching Windows Azure account that is mapped to your Windows LiveID. For our example, we chose `juggercloud` as the account name and received three storage endpoints. Two access keys are also generated.

Before we deploy our Web service, however, we will update the Web role service configuration `*.cscfg` file with the account name and account key information, as follows:

```
<ServiceConfiguration serviceName="StandardMoldHost"
  xmlns="http://schemas.microsoft.com/
  ServiceHosting/2008/10/ServiceConfiguration">
  <Role name="ServiceDemo_WebRole">
    <Instances count="2" />
    <ConfigurationSettings>
      <Setting name="DiagnosticsConnectionString"
        value="DefaultEndpointsProtocol=https;
        AccountName=standardmold;AccountKey=0lg820j...==" />
    </ConfigurationSettings>
  </Role>
</ServiceConfiguration>
```

Example 8.14

2. Create and Deploy a Service Package

We deploy the service by uploading a package through the Windows Azure portal. When using the Windows Azure UI, we can navigate to the host service to determine whether we are deploying to staging or production.

There's really no difference in hardware resource configuration between these two settings. In fact, the separation between the two environments is managed through the network load balancer's routing tables.

Once we click "Deploy," the package and configuration file will be uploaded.

NOTE

We could have also pulled these bits from a Windows Azure storage account. For example, we could create a custom MSBuild task leveraged within a Team Foundation Server Team Build definition file. Instead of dropping the package to a normal file drop, this would upload it into blob storage using the REST API, or perhaps even leverage Windows Azure Drive.

3. Promote the Service to Production

Let's imagine the previous step initially deployed the service to staging so that we could test it before moving it into the production environment. The Windows Azure UI allows you to invoke the service by clicking “Run,” resulting in a page similar to Figure 8.9.



Figure 8.9

After verifying that the Web service is performing as desired, it can be deployed to production (Figure 8.10).

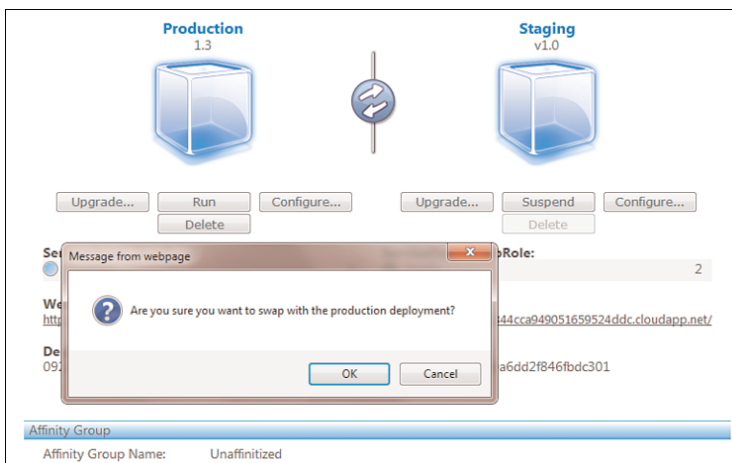


Figure 8.10

8.6 A REST Service in Windows Azure

In order to explore how REST services are created and exist within Windows Azure, this section takes the Web service from the previous section and makes it RESTful. But, before we dive into the implementation details of this change, let's first take a step back and think about REST-specific design considerations.

REST Service Addressing

A common design practice with REST services is to make the addressing (the manner in which target resources are addressed) as intuitive as possible. The social bookmarking site Delicious is a great example of this.

With Delicious, every bookmark has one or more tags (think of tags as categories). Tags essentially replace folders within Web browsers with categories. In relation to our discussion, you can also group tags into a bundle, which basically creates “tag clouds.” Access to tagged bookmarks is provided via REST services. Table 8.1 shows a set of sample URLs that can be used to get back a list of bookmarks for Azure, SOA, and Azure+SOA, respectively.

| URL | Description |
|---|--|
| http://delicious.com/tag/azure | returns a list of bookmarks that have been tagged with Azure |
| http://delicious.com/tag/soa | returns a list of bookmarks that have been tagged with SOA |
| http://delicious.com/tag/soa+azure | returns a list of bookmarks that have been tagged with SOA and Azure |

Table 8.1

Sample URLs used to retrieve different values from REST services at delicious.com.

What's important about this example is that we are able to search, create and update a large network of data via REST without writing code. The HTTP GET method and the appropriate URLs are all we need.

Returning to our Order service, we first need to define an appropriate resource addressing structure for the order data, as shown in Table 8.2.

| Action | IOrderService Operation Name | URI Address Template | HTTP Method |
|---|------------------------------|----------------------|-------------|
| get a list of all orders | GetOrders | ./orders | GET |
| get an order given the order ID | GetOrderByOrderId | ./order/{id} | GET |
| get a list of orders for a given customer | GetOrdersByCustomer | ./orders/{custName} | GET |
| create an order | CreateOrder | ./orders | POST |
| update an order | UpdateOrder | ./order/{id} | PUT |
| delete an order | DeleteOrder | ./order/{id} | DELETE |

Table 8.2

The resource addressing structure for the Order service.

Creating a Windows Azure REST Service

We now need to carry out a series of steps to make this a REST service:

1. Add a reference to `System.ServiceModel.Web` in the Contract project.
2. Add HTTP attributes to the methods defined in the `IOrderService` interface.
3. Update the WCF behavior.
4. Update the `OrderService.svc` file by adding a Web factory reference.

The `System.ServiceModel.Web` namespace contains classes that make up the Web HTTP programming model.

For our purposes, we need to focus on the following:

- `WebGetAttribute` (maps to an HTTP GET)
- `WebInvokeAttribute` (maps to HTTP POST, PUT, and DELETE)
- `WebMessageFormat` (defines the format of the response message)

For the GET method, we use the `WebGet` attribute. We then use the `UriTemplate` attribute to define the addressing structure from Table 8.2. This is a manual process, which means that it's easy to make mistakes. It is therefore important to lay out the URI structure prior to working with the code.

We also need to specify the {token} parameters. For example, if we were calling the `GetOrderByOrderId` operation of the Web service via SOAP, we would just pass in the order ID argument by calling the Web method. But with REST, everything is through HTTP methods and URIs. The service consumer doesn't call `GetOrderByOrderId` directly, but rather does the HTTP GET method on `http://server/OrderService.svc/order/2`, where "2" is the order ID value.

Next, we need to determine the response message format by setting `ResponseFormat` to return XML messages.

Here's what `IOrderService` looks like now:

```
[ServiceContract]
public interface IOrderService
{
    [WebInvoke(Method="POST",
        UriTemplate="orders",
        ResponseFormat=WebMessageFormat.Xml)]
    [OperationContract]
    int CreateOrder(Order o);
    [WebInvoke(Method="PUT",
        UriTemplate="order/{id}",
        ResponseFormat=WebMessageFormat.Xml)]
    [OperationContract]
    void UpdateOrder(string id, Order o);
    [WebGet(UriTemplate="order/{id}",
        ResponseFormat=WebMessageFormat.Xml)]
    [OperationContract]
    Order GetOrderByOrderId(string id);
    [WebGet(UriTemplate="orders/{custName}",
        ResponseFormat=WebMessageFormat.Xml)]
    [OperationContract]
    List<Order> GetOrdersByCustomer(string custName);
    [WebGet(UriTemplate="orders",
        ResponseFormat=WebMessageFormat.Xml)]
    [OperationContract]
    List<Order> GetOrders();
    [WebInvoke(Method="DELETE",
        UriTemplate="order/{id}",
        ResponseFormat=WebMessageFormat.Xml)]
    [OperationContract]
    void DeleteOrder(string id);
}
```

Example 8.15

We now need to update the WCF behavior in the Web.Config file by changing the endpoint binding to `WebHttpBinding` and the endpoint behavior to a `Web` behavior, as shown here:

```
<services>
  <servicebehaviorConfiguration=
    "ServiceDemo_WebRole.ServiceDemoBehavior"
    name="ServiceDemo_WebRole.OrderService">
    <endpoint address="" binding="WebHttpBinding"
      contract="Contract.IOrderService"
      behaviorConfiguration="Web">
    </endpoint>
  </service>
</services>
<behaviors>
  <endpointBehaviors>
    <behavior name="Web" />
  </endpointBehaviors>
  <serviceBehaviors>
    <behavior name=
      "ServiceDemo_WebRole.ServiceDemoBehavior">
    </serviceBehaviors>
  </behaviors>
```

Example 8.16

Finally, we have to update the `OrderService.svc` file to include `WebServiceHostFactory`, as shown here:

```
<%@
  ServiceHost Language="C#" Debug="true"
  Service="ServiceDemo_WebRole.OrderService"
  CodeBehind="OrderService.svc.cs"
  Factory="System.ServiceModel.
    Activation.WebServiceHostFactory"
%>
```

Example 8.17

`WebServiceHostFactory` provides instances of `WebServiceHost` in managed hosting environments, where the host instance is created dynamically in response to incoming messages. This is necessary because the service is being hosted using a Web role in IIS.

Finally, to deploy the REST version of the Order service to Windows Azure, we can follow the same steps described in the previous *A Web Service in Windows Azure* section.

SOA PRINCIPLES & PATTERNS

Cloud-based REST service architecture relates to several SOA design patterns relevant to REST service design. These are covered separately in the book *SOA with REST* as part of the *Prentice Hall Service-Oriented Computing Series* with Thomas Erl.

SUMMARY OF KEY POINTS

- Programming models and deployment processes for Web services are very similar and consistent between cloud-based services in Windows Azure and on-premise services in Windows Server.
- Most significant differences with cloud-based services in Windows Azure are managed via service configurations.
- Development and deployment of REST-based services in Windows Azure are also consistent with the on-premise platform.

8.7 Windows Azure Storage

Windows Azure provides the following set of storage services (collectively referred to as Windows Azure Storage), each of which is suitable for different types of data access requirements:

- *Tables* provide structured storage, as they do in regular databases. Essentially, each table consists of a set of data entities that each contain a set of properties.
- *Queues* provide reliable storage and delivery of messages. They are often used between roles to communicate with each other.
- *Blobs* are used to store large binary objects (files). They provide a simple interface for storing named files along with metadata and include support for CDN (Content Delivery Network).
- *Windows Azure Drives* provide durable NTFS volumes for Windows Azure applications.

Windows Azure Storage supplies a managed API and a REST API, both of which essentially provide the same level of functionality. The managed API is provided through the `Microsoft.WindowsAzure.StorageClient` namespace. To interact with the storage services, you can also use familiar programming interfaces, such as ADO.NET Data Services (available in the .NET framework version 3.5 SP1).

Note that access to storage is regulated via Windows Azure Storage accounts that use 256-bit secret keys. Also, there are some storage size limitations. For example, each storage account will have a maximum 100 terabytes of total storage capacity.

Tables

Windows Azure Tables (WATs) are similar to relational tables insofar as they both are used to store structured data. However, it's important to understand that WAT storage is not a relational database management system for the cloud (that's what SQL Azure is for). In other words, there is no support for common database features, such as joins, aggregates, stored procedures, or indexes.

WATs were built primarily to realize scalability, availability, and durability of data. Individual tables can be scaled to billions of entities (rows) with data totaling into the order of terabytes. Part of the scaling algorithm is that as application traffic and usage grows, WATs will automatically scale out to potentially tens, to hundreds, to thousands of servers. With regards to availability, each WAT is replicated at least three times.

Entities and Properties

Windows Azure Storage introduces some specific terminology and relationships for WATs:

- You create a *storage account*, each of which can have multiple *tables*.
- Data stored within a table is organized into *entities*. A database row is comparable to an entity.
- Each entity contains a set of *properties*. A database column is comparable to a property.
- A table is comprised of a set of entities, each of which is comprised of a set of properties.

Each entity contains two key properties that together form the unique ID of the entity in that table. The first key is the *PartitionKey*, which allows you to group entities together.

This tells the Windows Azure Storage system to not split this group up when scaling out the table.

In other words, partition keys are used to group table entities into partitions that provide a unit of scale that Windows Azure Storage uses to properly load balance data. Partition keys also allow you to control the physical locality of the entity data. Everything within a partition will live on a single server.

The second key is the *RowKey*, which provides uniqueness within a partition in that the *PartitionKey* together with the *RowKey* uniquely identify a given entity (as well as the sort order). You can think of these two keys as a clustered index for a table.

The third required attribute is the *Timestamp*, which is a read-only attribute used to control optimistic concurrency. That is, if you try to update a row that another program has already updated, your update attempt will fail because of the timestamp mismatch.

Data Access

When interacting with entities and properties, you are provided the full range of regular data access functions (get, insert, update, delete), in addition to special features, such as the partial update (merge), the entire update (replace), and the entity group transaction.

Entity group transactions allow you to atomically perform multiple insert, update, and delete commands over a set of entities in the same partition as part of a single transaction.

Queues

As with traditional messaging queues, the Windows Azure queues provide a reliable intermediary mechanism for delivering messages. For example, a common scenario is to set up a queue as the communication proxy between an application's Web role (of which there may be one or two instances) and its worker roles (of which there can be many instances). For this scenario you would likely set up at least two queues. The first would allow the Web role to submit messages for the worker roles to process. The worker roles would poll the queue for new messages until one is received. The second queue would then be for the worker roles to communicate back to the Web role. This architecture allows the Web role to delegate and spread out resource-intensive work to the worker roles.

Just like with tables, queues are scoped by the storage account that you create. An account can have many queues, each of which can contain an unlimited amount of

messages. Also, dequeued counts are tracked, allowing you to determine how often a given message has been dequeued by a worker process.

Queues offer a range of data access functions, including the ability to create, delete, list, and get/set queued metadata. Additionally, you can add (enqueue) and retrieve (dequeue) sets of messages, and delete and “peek” at messages individually.

Blobs

Each storage account can have containers that can be used to store blobs. There is no limit to the number of containers that you can have as long as they will fit into your storage account limit.

Containers have the ability to set public or private access policies. The private access level will only allow access to consumers that have been given permission. Public access allows any consumer to interact with the container’s blobs using a URL. You can also have container metadata, which, like blob metadata, is stored in name-value pairs.

You have two choices for the type of blob that you can use: block and page. Both types have characteristics that make them applicable to specific requirements.

Block Blobs

A block blob is primarily geared towards streaming media files. Each blob is organized into a sequential list of “blocks” that can be created and uploaded out of order and in parallel for increased performance. Once uploaded, each block is in an uncommitted state, meaning that you cannot access the blob until its blocks are committed. To commit the blocks as well as define the correct block order, you use the `PutBlobList` command.

Each block is immutable and is further defined by a block ID. After you have successfully uploaded a block, that block (identified by its block ID) cannot be changed. That also means that if you have updated a block on-premise, then you will need to re-upload or copy the entire block with the same block ID.

Blobs can be accessed via an available REST API that provides standard data access operations, as well as special functions, such as *CopyBlob* that allows you to copy an existing blob to a new blob name.

Page Blobs

Page blobs are suitable for random I/O operations. With this kind of blob, you must first pre-allocate space (up to 1TB), wherein the blob is divided into 512-byte “pages.” To access or update a page, you must address it using a byte offset. Another key difference is that changes to page blobs are immediate.

You can expand the blob size at any point by increasing its maximum size setting. You are also allowed to shrink the blob by truncating pages. You can update a page in one of two ways: *PutPage* or *ClearPage*. With *PutPage*, you specify the payload and the range of pages, whereas *ClearPage* basically zeroes out a page range up to the entire blob. There are several other commands that can be used to work with page blobs.

Windows Azure Drive

Windows Azure Drive is a storage service that provides a durable NTFS volume for Windows Azure applications. An application needs to mount the volume prior to using it and, when done, the application then unmounts the same volume. Throughout this period, the volume data is kept intact, even if the application should crash.

A Windows Azure Drive volume is actually a page blob. Specifically, it exists as a page blob that has been formatted as an NTFS single volume virtual hard drive (VHD). As such, these drives can be up to 1TB in size.

SUMMARY OF KEY POINTS

- Windows Azure Storage provides a set of services for distributed cloud-based data storage.
 - The four types of storage services provided are tables, queues, blobs, and Windows Azure Drive.
 - Windows Azure Storage services are available via both .NET managed APIs and REST-based APIs.
-

Index

A

Abort() method

(**ICommunicationObject**), 814

abstracted APIs (**BizTalk BAM**),
666-669

Access Control, 548-553

cross-domain access control in

cloud computing, 544

service authorization scenarios,
553-554

Standard Mold case study, 555-556

in Windows Azure, 528

access tokens, requesting, 551

activation types in **.NET Remoting**,
58, 60

Active Directory Federation Services
(**ADFS**), 539

activities (in **WF**), 172-175, 662

ad hoc discovery, 141, 143-144

adapter providers (**ESB Toolkit**),
478-479

adapters (**BizTalk Server**), 199-200

address attribute (service endpoints),
88-89

ADFS (Active Directory Federation
Services), 539

administration management tools
(**WCF**), 151

Agnostic Capability design pattern,
169, 334, 709

Agnostic Context design pattern, 169,
182, 332-333, 377, 710

agnostic logic, 32

Agnostic Sub-Controller design
pattern, 711

alerts, 165, 670-671

ambient transactions, 69

analysis, inventory analysis cycle,
337-338

Annotated SOA Manifesto, 796-808

AppFabric. *See* Windows Azure
platform AppFabric, Windows
Server AppFabric

Application Blocks, 182-183

application container (Windows
Azure), 216-217. *See also* hosting
models

application domains, 109

application integration models,
467-470

ASMX (XML Web services), 71, 73

code generation, 302-304

industry standards supported by, 70

programming model for **WF**

orchestrations, 397

publishing workflows as, 399-407

ASP.NET, caching REST responses,
599-601

assemblies, history of, 51

asynchronous interactions, 639-641

Asynchronous Queuing design
pattern, 712

Atomic Service Transaction design
pattern, 127, 388, 435, 458, 713

attachments (SOAP), 701

authentication. *See also* Brokered
Authentication design pattern;
Direct Authentication design pattern
brokered authentication, 519
defined, 122
direct authentication, 518-519
mutual authentication, 520
Service Bus security, 508-509

authorization
claims-based authorization,
524-529, 546-547
defined, 122
role-based authorization, 520-524
service authorization scenarios
(Access Control), 553-554
WCF security, 125-126

AuthorizationContext class, 527

Azure. *See* Windows Azure

B

backup lists, 139

BAM (Business Activity
Monitoring), 194

abstracted APIs, 666-669
capturing data, 655-656
EventStream APIs, 665-666
interceptors, 670
management
 database outages, 672
 reporting, 676

scripting deployment, 673-676

security, 672-673

notifications, 670-671

rapid prototyping, 671

real-time versus scheduled

 aggregations, 660-661

service composition metrics, 669

solution architecture, 656-659

Standard Mold case study, 677-680

Tracking Profile Editor (TPE),
659-660

BAM Management utility
(BM.EXE), 659

base activity library (WF), 174

behaviors

 defining for routing services, 136
 WCF, 104-105

binding attribute (service endpoints),
89-92

bindings, cleaning up resources, 822

BizTalk BAM. *See* BAM (Business
Activity Monitoring)

BizTalk Operations service (ESB
Toolkit), 471

BizTalk Server, 193

adapters, 199-200

architecture, 193-195

context properties, 200-201

ESB Toolkit. *See* ESB Toolkit

hosts, 481

hub-bus integration model, 469-470

messaging, 193, 196

pipelines, 197-198

ports and locations, 199

orchestration and, 193, 388-391,
443-445

Compensating Service

Transaction design pattern,
456-461

- exception management*, 202-203
- itineraries*, 201
- orchestrated task service contracts*, 445-447
- Process Centralization design pattern*, 450-451, 454
- State Repository design pattern*, 455-456
- Superior Stamping case study*, 448-449
- WS-* support*, 447-448
- parallelism in, 643-644
- productivity tools in, 194
- request throttling, 607-608
- selecting hosting models, 611-612
- blobs in Windows Azure Storage**, 239, 242-243
- block blobs in Windows Azure Storage**, 242
- BM.EXE (BAM Management utility)**, 659
- books, related to this book**, 4-6, 41-42
- bootstrapper**, 574
- brokered authentication**
 - mechanism, 519
- Brokered Authentication design pattern**, 126, 714
- BufferedEventStream API (BizTalk BAM)**, 665
- Business Activity Monitoring. See BAM (Business Activity Monitoring)**
- business activity services (BizTalk Server)**, 194
- business metrics**, 655
- business processes, human intervention in**, 450-451
- business rules in Windows Workflow Foundation (WF)**, 180-181
- business rules engine (BizTalk Server)**, 194
- C**
- cache hit ratio**, 601
- cache keys, computing**, 593-594
- caching**, 587-591
 - cache keys, computing, 593-594
 - implementation technologies, 592-593
 - in-memory application cache (Windows Server AppFabric), 190-191
 - intercepting messages, 589
 - at caching utility service, 590
 - comparison of techniques, 591
 - at intermediary, 589
 - at service proxy, 590
 - in service container, 589
 - monitoring cache efficiency, 601
 - REST responses, 599-601
 - Standard Mold case study, 594-596
 - Superior Stamping case study, 597-599
- caching utility service, intercepting messages**, 590
- Canonical Expression design pattern**, 78, 147, 280
 - profile, 715
 - service capability naming conventions, 281-282
 - service naming conventions, 280
- Canonical Protocol design pattern**, 274
 - component implementation, 278-279
 - dual protocols, 279-280

- named pipes, 279
- profile, 716
- REST services implementation, 277-278
- Web services implementation, 275-277
- Canonical Resources design pattern, 249, 717**
- Canonical Schema Bus compound pattern, 719**
- Canonical Schema design pattern, 78, 82, 84, 217, 251, 253, 404, 445-446, 638**
 - creating schemas, 254, 256-258
 - DataContract library, 264-267
 - generating .NET types, 258-264
 - profile, 718
- Canonical Versioning design pattern, 720**
- CAO (client-activated objects), 58**
- Capability Composition design pattern, 374-375, 721**
- capability granularity, 38, 308-313**
- Capability Recomposition design pattern, 374-376, 722**
- capturing data, 655-656**
- case studies background, 16-19. *See also* code examples**
- case studies conclusion, 686. *See also* code examples**
- channel bindings, 148**
- channel layer, 149**
- channel layer extensibility (WCF), 150**
- channel stack, 149**
- ChannelFactory class, 119-120, 815**
- chapters, described, 6, 8-11**
- chatty interface, 311**
- claim sets, 526**
- claims, 534, 546**
 - accessing, 527
 - designing custom, 529
 - generating, 525
 - passport example, 525
 - processing, 526
- claims-based authorization, 524-529, 546-547**
- ClaimSet class, 527**
- classes in .NET Remoting, 56**
- cleaning up resources, 811-812**
 - Abort() method (ICommunicationObject), 814
 - based on bindings, 822
 - ClientBase and ChannelFactory classes, 815
 - Close() method (ICommunicationObject), 812-814
 - Dispose() method (IDisposable), 814-815
 - try-catch-finally-abort block, 817
 - try-close-catch-abort block, 818-819
 - using block, 816
 - utility methods, 819, 821
- client connections, closing, 822**
- client-activated objects (CAO), 58**
- client-server architecture, 44-45**
- client-side hybrid itineraries, 474**
- client-side itineraries, 474**
- ClientBase class, cleaning up resources, 815**
- clients, 537**
- Close() method (ICommunicationObject), 812-814**

closing

- client connections, 822
- database connections, 812
- ICommunicationObject. *See*
cleaning up resources

cloud computing. *See also***Windows Azure**

- cloud deployment models, 208
 - community cloud*, 209
 - dedicated cloud*, 209
 - hybrid cloud*, 209
 - intercloud*, 209
 - private cloud*, 208
 - public cloud*, 208
 - Windows Azure and*, 210
- ESB with Windows Azure, 483
 - receiving messages from Service Bus*, 484-485
 - sending messages to Service Bus*, 485-486
- explained, 206-208
- security, 543-547
- service delivery models, 210-212

Cloud Computing & SOA*, 6*cloud service projects, creating (Hello World example), 224****cloud-to-cloud authorization model (Access Control), 554****co-hosting, 645-648****coarse-grained service contracts, 608-610****code examples**

- accessing claims, 527
- activities calling REST services, 424
- adding discovery endpoints, 142
- adding services to runtime, 414
- address attribute (service endpoints), 88

ambient transactions, 69**ASP.NET role provider****authentication, 523****AspNetCacheProfile attribute, 599****aspNetCompatibilityEnabled attribute, 600****AspNetCompatibilityRequirements attribute, 600****asynchronous interactions, 640-641****asynchronous interfaces for****workflows, 402****authorization, 126****backup lists, 139****behaviors (WCF), 105****binding configuration, 418****binding workflows to WCF contracts, 409****bootstrapper setup, 574-575****bootstrapping workflow****instances, 424****cache creation in Windows Server****AppFabric, 191****caching (Standard Mold case study), 595-596****caching (Superior Stamping case study), 597-599****caching profile, 601****channel layer extensibility, 150****ChannelFactory class, 119-120****claim processing, 526****client-activated objects, 60****CLR type serialization, 83****co-hosting, 647****coarse-grained service interface, 312****communication protocols in .NET****Remoting, 61****component implementation, 279****configuration file contents (ASMX services), 407**

- configuring REST endpoints, 421
- configuring Send/Receive activities
 - for REST services, 423
- configuring service behaviors, 432
- consuming RESTful services
 - directly, 96
- contract-first development, 411
- converting service operations, 94
- creating filter tables for routing services, 138
- creating message queues, 64
- custom discovery metadata, 143
- data model transformations, 269
- DataContract library, 266
- declarative authorization, 524
- defining behaviors for routing services, 136
- defining endpoints for routing services, 136
- defining message filters for routing services, 137
- defining target endpoints for routing services, 137
- defining workflow contract, 416
- deleting message queues, 65
- development-time deployment script, 674
- development-time undeployment script, 675
- Diagnostic Monitor, 229
- Discovery Proxies, 146
- documentation (Standard Mold case study), 351
- durable services, 131
- dynamically loading modules, 580-581
- elements in SOAP messages, 413
- endpoint element, 87
- exception shielding, 319-321
- exceptions for MSMQ message processing, 66
- exporting BizTalk orchestrations to WS-BPEL, 452
- façade creation, 306-307
- fault contracts, 99-100
- filter rules referencing backup lists, 140
- fine-grained component interface, 310
- Flexible Strategy (versioning), 103
- GUID extraction (Standard Mold case study), 345
- high performance transports (Standard Mold case study), 627
- IIIdentity interface, 522
- IIS hosting, 113-114
- imperative authorization, 524
- implicit programming model (System.Transactions library), 69
- implicit service discovery, 147-148
- importing types into XML schemas, 289
- in-line documentation, 322
- interface for request-response operation, 400
- internal endpoint definition for IRC, 223
- internal endpoint listings for IRC, 223
- internal endpoint specification for IRC, 222
- IOrderService in REST services, 237
- IOrderService interface
 - contract, 231

- IPrincipal interface, 521
- JSON encoding, configuring
 - contracts for, 422
- launching svcutil, 294
- LINQ-to-XML mapping, 272
- listener configuration for service
 - announcements, 145
- logging and diagnostic APIs, 229
- managed Windows services, 112
- meeting notes (Standard Mold case study), 363
- message contracts, 85
- message contracts, changing
 - parameters, 85
- message logging, 153
- message schema, 291
- messaging queues (Standard Mold case study), 348-350
- module creation, 565, 575-577
- MTOM encoding, 628
- MTOM encoding (Standard Mold case study), 629
- multiple service endpoints, 87
- namespaces in generated services, 404
- .NET type generation, 264
- .NET API interaction (Standard Mold case study), 341
- notification properties (Standard Mold case study), 347
- object-to-object mapping, 271
- operation attributes for
 - transactions, 128
- orchestration and BizTalk Server (Superior Stamping case study), 449
- Order data contract, 231
- output of IOrderService interface
 - contract and Order data contract, 232
- passing data to workflows, 416
- passing parameters to workflow instances, 178
- performance policies (Standard Mold case study), 621
- persistence service parameter list, 430
- persistence services, attaching to
 - workflow runtime, 428
- probe queries, 144
- probe request management, 142
- production deployment script, 674
- production undeployment script, 675-676
- Query Notification (SQL Server), 165
- raw messages in SOAP, 412
- receiving MSMQ messages, 66
- region/view association, 578
- regions in shells, 564
- relying parties, developing, 542
- request throttling (Superior Stamping case study), 607
- request throttling with WCF, 605
- REST-based Service Bus
 - contracts, 514
- REST contracts in WCF, 420
- REST service creation (Standard Mold case study), 355-356
- REST service hosting, 116
- REST service implementation, 278
- REST service XML output (Standard Mold case study), 358
- retrieving XML (Standard Mold case study), 364

- returning data from services, 82
- returning parameters from
 - workflow instances, 178-179
- RoutingService class, 133-134
- saving messages (Standard Mold case study), 346
- self-hosted services, 111-112
- sending MSMQ messages, 65
- sending service announcements, 144
- service attributes for
 - transactions, 129
- Service Broker (SSB)
 - implementation, 163-164
- Service Bus configuration file, 510
- Service Bus contract definition, 506
- Service Bus contract
 - implementation, 507
- Service Bus message buffer
 - creation, 516
- Service Bus service
 - configuration, 508
- Service Bus service host
 - configuration, 512
- Service Bus service publication, 512
- service consumer code creation (Standard Mold case study), 359-360, 362
- service consumer code XML output (Standard Mold case study), 361, 363
- service contract behavior
 - attributes, 80
- service contract creation, 80
- service contract definition, 79
- service contract design (Standard Mold case study), 631-632
- service contract implementation, 107-108
- service contracts (Standard Mold case study), 628
- service contracts, changing default name and namespace, 80
- service definition (ASMX services), 403
- service endpoint attributes, 81
- Service Metadata Tool usage, 117
- ServiceConfiguration.cscfg, 228
- ServiceDefinition.csdef, 228
- ServiceFault data contract, 99
- ServiceHost class, 109
- shell control setup, 573-574
- shell implementation, 561-562
- signing messages (Standard Mold case study), 620
- SingleCall objects, 59
- Singleton objects, 59
- SOAP messages, 85-86
- SQL Server endpoint creation, 158
- SQL Server stored procedure
 - creation, 158
- Standard Mold case study metrics, 678-679
- stateful interactions, 131
- .svc file for compiled workflow, 418
- .svc file for REST contracts, 421
- svcutil-generated code, 296
- throttling controls, 105
- timestamps (Standard Mold case study), 365
- tracing, 152
- tracing listener creation, 230
- TrackServiceEnd method, 668
- TrackServiceException method, 668
- TrackServiceStart method
 - overload, 667
- transactions, 53

- TransactionTimeout attribute, 130
- try-catch block for fault contracts, 100
- try-catch blocks, 98
- updating configuration settings, 230
- updating database information (Standard Mold case study), 366-367
- updating OrderService.svc file for REST services, 238
- updating WCF behavior in REST services, 238
- updating Web role service configuration file, 233
- UriTemplate attribute (REST services), 96-98
- validation abstraction, 316-318
- verifying metadata compatibility, 146
- view creation, 577
- view definition, 562
- WCF interface (Standard Mold case study), 344
- WCF security (Standard Mold case study), 532-533
- Web service implementation, 276-277
- WebGet attribute (REST services), 93
- WebInvoke attribute (REST services), 95
- WebRole.cs, 227
- WebService attribute, 72
- @WebService directive, 71
- WF programming model, 177
- Windows Authentication binding for a service, 522
- workflow host associations, 432
- workflow persistence (with AppFabric), 190
- workflow persistence (with WF), 171
- workflows as ASMX services, 404
- writing proxy class, 118-119
- WSCF.blue-generated code, 301
- WSCF.classic-generated code, 304
- WSDL definition, 293
- XML schema creation with Visual Studio, 255-257
- XSLT transformation, 273
- COM, history of, 48-49**
- COM+ services**
 - history of, 49, 51
 - hosting .NET Remoting components, 57
- Command design pattern, 568**
- Common Object Request Broker Architecture (CORBA), 47**
- communication protocols**
 - in .NET Remoting, 60
- communication with host container, 171-172**
- community cloud deployment model, 209**
- Compatible Change design pattern, 102, 316, 723**
- Compensating Service Transaction design pattern, 132, 387, 434**
 - BizTalk Server, 456-461
 - creating compensations, 434-435
 - profile, 724
 - triggering compensations, 435
- components**
 - as service implementation option, 278-279
 - services as, 29
- Composite Application Library for WPF and Silverlight. *See* Prism Library**
- Composite design pattern, 568**

- Composite View design pattern**, 568
- composite views (Prism Library)**, 562
- composition**. *See* **service composition**
- Composition Autonomy design pattern**, 645, 725
- composition controller**, 377
- composition controller capabilities**,
 - design characteristics**, 704-705
- composition initiator**, 377
- composition member**, 377
- composition member capabilities**,
 - design characteristics**, 704
- composition sub-controller**, 377
- compound patterns**. *See also* **design patterns**
 - Canonical Schema Bus, 719
 - Enterprise Service Bus, 136, 194, 219. *See also* ESB (enterprise service bus)
 - mapping ESB to*, 490, 492
 - profile*, 741
 - Federated Endpoint Layer, 249, 745
 - Official Endpoint, 757
 - Orchestration, 169, 194, 382, 395, 443, 758
 - Service Broker, 136, 160-164, 771
 - Three-Layer Inventory, 378, 788
- Concurrent Contracts design pattern**, 102, 226, 307-308, 726
- confidentiality, defined**, 122
- configuration**
 - router services, 135-139
 - Service Bus, 504-512
 - authentication*, 508-509
 - contract definition*, 505-506
 - contract implementation*, 506-507
 - end-to-end security*, 509
 - service configuration*, 508
 - service host configuration*, 511-512
 - service publication*, 512
 - SQL Express, 224
- connections, closing**
 - client connections, 822
 - database connections, 812
- connectivity fabric, Service Bus as**, 494-496
- connectivity models (Service Bus)**, 499
 - Eventing, 500
 - Service Remoting, 501
 - Tunneling, 501-502
- console applications, hosting .NET Remoting components**, 57
- constraint granularity, defined**, 38
- consumer-to-contract coupling**, 286
- consumer-to-implementation coupling**, 286
- consumer-to-logic coupling**, 310
- consumers**. *See* **service consumers (WCF)**
- contention**, 603-604
- context properties (BizTalk Server)**, 200-201
- context property**, 660
- contract attribute (service endpoints)**, 92
- Contract Centralization design pattern**, 286, 288, 727
- Contract Denormalization design pattern**, 313, 630, 728
- contract-first development**, 250-252, 694
- contract-to-functional coupling**, 286
- contract-to-implementation coupling**, 286
- contract-to-logic coupling**, 285

- contract-to-technology coupling, 286
- conversation group identifier, 161
- conversation groups, 161
- CORBA (Common Object Request Broker Architecture), 47
- coupling. *See* Service Loose Coupling design principle
- CREATEENDPOINT statement (SQL Server), 158-159
- credentials, 123-125
- cross-domain access control in cloud computing, 544
- Cross-Domain Utility Layer design pattern, 182, 729
- custom encryption, 623-625

D

- data access in Windows Azure Storage, 241
- Data Confidentiality design pattern, 125, 730
- data contracts
 - creating, 251
 - DataContract attribute, 82-83
 - defined, 78
 - fault contracts, 98-100
 - versioning, 102
- data dimension, 664
- Data Format Transformation design pattern, 200, 731
- data granularity, 38
- Data Model Transformation design pattern, 200, 267-269, 388, 446, 453, 637
 - LINQ-to-XML mapping, 271-272
 - object-to-object mapping, 269-271
 - profile, 732
 - XSLT transformations, 272-273

- Data Origin Authentication design pattern, 125, 733
- data serialization, 82-83
- database connections, closing, 812
- database outages, BizTalk BAM management, 672
- databases. *See* SQL Azure; SQL Server
- DataContract attribute (WCF), 82-83
- DataContract library, 264-267
- DCOM (Distributed Component Object Model), 47-49
- declarative authorization, 524
- Decomposed Capability design pattern, 734
- Decoupled Contract design pattern, 81, 207, 217, 277, 288, 397, 399
 - profile, 735
 - svcutil utility program, code generation with, 294-297
 - WSCF.blue, code generation with, 297-301
 - WSCF.classic, code generation with, 302-304
 - WSDL-first approach, 289-290, 292-294
- dedicated cloud deployment model, 209
- dependency injection container, 566, 571
- Dependency Injection design pattern, 570-571
- deployment models in cloud computing, 208
 - community cloud, 209
 - dedicated cloud, 209
 - hybrid cloud, 209
 - intercloud, 209
 - private cloud, 208
 - public cloud, 208
 - Windows Azure and, 210

deployment scripting, BizTalk BAM management, 673-676

design patterns. *See also* compound patterns

- Agnostic Capability, 169, 334, 709
- Agnostic Context, 169, 182, 332-333, 377, 710
- Agnostic Sub-Controller, 711
- Asynchronous Queuing, 712
- Atomic Service Transaction, 127, 388, 435, 458, 713
- Brokered Authentication, 126, 714
- Canonical Expression, 78, 147, 280
 - profile, 715*
 - service capability naming conventions, 281-282*
 - service naming conventions, 280*
- Canonical Protocol, 274
 - component implementation, 278-279*
 - dual protocols, 279-280*
 - named pipes, 279*
 - profile, 716*
 - REST services implementation, 277-278*
 - Web services implementation, 275-277*
- Canonical Resources, 249, 717
- Canonical Schema, 78, 82, 84, 217, 251, 253, 404, 445-446, 638
 - creating schemas, 254, 256-258*
 - DataContract library, 264-267*
 - generating .NET types, 258-264*
 - profile, 718*
- Canonical Versioning, 720
- Capability Composition, 374-375, 721

- Capability Recomposition, 374-376, 722
- Command, 568
- Compatible Change, 102, 316, 723
- Compensating Service Transaction, 132, 387, 434
 - BizTalk Server, 456-461*
 - creating compensations, 434-435*
 - profile, 724*
 - triggering compensations, 435*
- Composite, 568
- Composite View, 568
- Composition Autonomy, 645, 725
- Concurrent Contracts, 102, 226, 307-308, 726
- Contract Centralization, 286, 288, 727
- Contract Denormalization, 313, 630, 728
- Cross-Domain Utility Layer, 182, 729
- Data Confidentiality, 125, 730
- Data Format Transformation, 200, 731
- Data Model Transformation, 200, 267-269, 388, 446, 453, 637
 - LINQ-to-XML mapping, 271-272*
 - object-to-object mapping, 269-271*
 - profile, 732*
 - XSLT transformations, 272-273*
- Data Origin Authentication, 125, 733
- Decomposed Capability, 734

- Decoupled Contract, 81, 207, 217, 277, 288, 397, 399
 - profile*, 735
 - svcutil utility program, code generation with*, 294-297
 - WSCF.blue, code generation with*, 297-301
 - WSCF.classic, code generation with*, 302-304
 - WSDL-first approach*, 289-290, 292-294
- defined, 38-39
- Dependency Injection, 570-571
- Direct Authentication, 736
- Distributed Capability, 737
- Domain Inventory, 182, 337, 371, 738
- Dual Protocols, 226, 279-280, 399, 625, 648, 739
- Enterprise Inventory, 740
- Entity Abstraction, 169, 336, 377-378
 - with .NET REST service (Standard Mold case study)*, 351-367
 - profile*, 742
- Event Aggregator, 570
- Event-Driven Messaging, 165, 445, 743
- Exception Shielding, 319-321, 529, 744
- File Gateway, 746
- Functional Decomposition, 169, 330-331, 445, 560, 747
- Intermediate Routing, 133, 136, 748
- inventory boundary patterns. *See* Domain Inventory design pattern
- Inventory Endpoint, 749
- inventory standardization patterns.
 - See* Canonical Protocol design pattern; Canonical Schema design pattern
- inventory structure patterns
 - Logic Centralization design pattern*, 751
 - Service Layers design pattern*, 169, 377-378, 779
 - Service Normalization design pattern*, 781
- Inversion of Control, 570
- Legacy Wrapper, 750
- Logic Centralization, 751
- Message Screening, 529, 752
- Messaging Metadata, 84, 201, 251, 753
- Metadata Centralization, 147, 219, 325-326, 417, 754
- Model-View-Controller, 568
- Model-View-Presenter, 568
- Model-View-View-Model, 568
- modularity design patterns, 569-571
- Multi-Channel Endpoint, 633, 755
- Non-Agnostic Context, 169, 379-380, 756
- Partial State Deferral, 456, 759
- Partial Validation, 760
- Passive View, 568
- Plug-In, 570
- Policy Centralization, 761

- for presentation layer, 567
 - modularity patterns*, 569-571
 - user interface patterns*, 567-569
- Presentation Model, 568-569
- Process Abstraction, 169, 378, 380, 382-385, 762
- Process Centralization, 169, 382-385, 425, 450
 - human intervention in business processes*, 450-451
 - profile*, 763
 - workflow design in WF*, 425-426
 - WS-BPEL support*, 426, 451, 454
- Protocol Bridging, 136, 200, 274, 764
- Proxy Capability, 765
- Redundant Implementation, 207, 213, 766
- reference notation, 12
- Reliable Messaging, 160, 767
- Rules Centralization, 181, 388, 768
- Schema Centralization, 217, 251, 253, 445, 769
- Separated Interface, 570
- Separated Presentation, 568-569
- Service Agent, 133, 589, 770
- Service Callback, 772
- Service Data Replication, 190, 218, 773
- Service Decomposition, 774
- Service Encapsulation, 169, 332, 775
- Service Façade, 304, 306-307, 776
- Service Grid, 777
- Service Instance Routing, 778
- Service Layers, 169, 377-378, 779
- Service Locator, 571
- Service Messaging, 84, 780
- Service Normalization, 781
- Service Perimeter Guard, 529, 782
- Service Refactoring, 217, 783
- sources for information, 11-12
- State Messaging, 456, 784
- State Repository, 171, 207, 386, 426-428
 - BizTalk Server*, 455-456
 - persistence service and scaling out in WF 3.0*, 429-431
 - persistence service and scaling out in WF 4.0*, 431-433
 - profile*, 785
- Stateful Services, 190, 207, 213, 456, 590, 786
- Supervising, 568
- Supervising Presenter, 569
- Termination Notification, 102, 787
- Trusted Subsystem, 789
- UI Mediator, 568, 790
- Uniform Contract, 282
- user interface design patterns, 567-569
- Utility Abstraction, 169, 182, 335, 377-378
 - profile*, 791
 - with .NET Web service (Standard Mold case study)*, 339-351
- Validation Abstraction, 82, 217, 315-318, 792
- Version Identification, 102, 793
- design principles**
 - list of, 26-27
 - reference notation, 12
 - Service Abstraction, 26, 82, 84, 102, 104, 207, 217, 313-314, 696

- Service Autonomy, 27, 110, 190, 218, 699
- Service Composability, 27, 125, 373, 560, 704-705
- Service Discoverability, 27, 147, 219, 321, 417, 535
 - implementation*
 - requirements*, 703
 - in-line documentation*, 322
 - profile*, 702-703
 - REST services*, 323
 - service profiles*, 323-324
- Service Loose Coupling, 26, 81-82, 84, 217, 285-286, 559
 - design patterns and*, 286
 - profile*, 695
 - service capability granularity and*, 308-313
- Service Reusability, 26, 125, 329-330, 337, 339, 697-698, 704
- Service Statelessness, 27, 171, 190, 700-701
- sources for information, 11-12
- Standardized Service Contract, 26, 78, 81-82, 84, 217, 250, 274, 340, 399
 - contract-first development*, 250-252
 - design patterns and*, 252
 - profile*, 693-694
- dialogs**, 161
- digital identity**, 534-536, 546
- dimensions**
 - in real-time aggregations, 662
 - types of, 664
- direct authentication**, 518-519
- Direct Authentication design pattern**, 736
- direct connectivity model (Service Bus)**, 494-496
- DirectEventStream API (BizTalk BAM)**, 665
- discovering services**, WCF Discovery, 140-141. *See also* Service Discoverability design principle
 - discovery modes, 141-143
 - Discovery Proxies, 146
 - implicit service discovery, 147-148
 - probe queries, 143-144
 - sending/receiving service announcements, 144-146
- discovery modes**, 141-143
- Discovery Proxies**, 146
- Dispose() method (IDisposable)**, 814-815
- distributed architecture**, explained, 45-47
- Distributed Capability design pattern**, 737
- Distributed Component Object Model (DCOM)**, 47
- distributed computing**
 - client-server architecture, 44-45
 - distributed architecture, 45-47
 - service-oriented architecture, 47
- distributed ESBs**, 482-483
- distributed garbage collection**, 49
- distributed resource transactions**, 67-68
- Distributed Transaction Coordinator (DTC)**, history of, 52-53
- distributed transactions**, explained, 51
- documentation**, in-line, 322
- Domain Inventory design pattern**, 182, 337, 371, 738
- domain service inventory**, 34

DTC (Distributed Transaction Coordinator), history of, 52-53
 Dual Protocols design pattern, 226, 279-280, 399, 625, 648, 739
 durable services, WCF transactions, 131-132
 dynamically loading user-interface modules, 579-581

E

elevated mode (Visual Studio), 223
 encoding, MTOM, 627-629
 encryption, 622-625
 end-to-end security (Service Bus), 509
 endpoint element, 86-87
 address attribute, 88-89
 binding attribute, 89-92
 contract attribute, 92
 endpoints
 defining for routing services, 135-136
 SQL Server, creating, 158-160
 Enterprise Service Bus compound pattern, 136, 194, 219. *See also* ESB (enterprise service bus)
 mapping ESB to, 490, 492
 profile, 741
 enterprise single sign-on (BizTalk Server), 194
 Enterprise Inventory design pattern, 740
 entities in Windows Azure Storage, 240-241
 Entity Abstraction design pattern, 169, 336, 377-378
 with .NET REST service (Standard Mold case study), 351-367
 profile, 742

entity group transactions in Windows Azure Storage, 241
 entity service model, defined, 32
 errors, fault contracts, 98-100
 ESB (enterprise service bus), 466
 cloud-enabling with Windows Azure, 483
 receiving messages from Service Bus, 484-485
 sending messages to Service Bus, 485-486
 governance, 487
 monitoring, 488-489
 SLA enforcement, 488
 transition plans, 489
 high-availability architecture, 480
 distributed ESBs, 482-483
 scaling, 481-482
 mapping to Enterprise Service Bus compound pattern, 490, 492
 Microsoft and, 466-467
 ESB Guidance, 467
 ESB Toolkit, 470-471
 adapter providers, 478-479
 itineraries, 472-474
 lifecycle of, 475-476
 types of, 474
 resolvers, 476-478
 services provided by, 471
 Event Aggregator design pattern, 570
 Event Collector Service (Windows Server AppFabric), 192
 event notifications, 165, 670-671
 event sources, types of, 660
 Event-Driven Messaging design pattern, 165, 445, 743
 Eventing connectivity model (Service Bus), 500

- EventStream APIs (BizTalk BAM), 665-666
- exception management (BizTalk Server), 202-203
- Exception Management service (ESB Toolkit), 471
- Exception Shielding design pattern, 319-321, 529, 744
- explicit programming model (System.Transactions library), 68-69
- extensibility
 - WCF
 - channel bindings*, 148
 - channel layer extensibility*, 150
 - layers*, 149-150
 - Windows Workflow Foundation (WF), 180
- Extensible Application Markup Language (XAML), 560
- external cloud, 208
- external identity providers, 546
- ExternalDataExchange services, publishing workflows via, 413-416

F

- failover (in Windows Azure), 221
- fault contracts, 98-100
- fault tolerance, router services, 139-140
- FaultContract attribute (WCF), 98-100
- faults, Exception Shielding design pattern, 319-321
- Federated Endpoint Layer compound pattern, 249, 745
- federated ESBs, 482-483
- federated identity, WCF security, 126
- federation, 40
- File Gateway design pattern, 746

- filter tables, creating for routing services, 138-139
- Flexible strategy (versioning), 102
- Functional Decomposition design pattern, 169, 330-331, 445, 560, 747

G

- GAT (Guidance Automation Toolkit), 184
- GAX (Guidance Automation Extensions), 184
- glossary Web site, 13, 810
- governance, ESB, 487-489
- granularity
 - capability granularity, 308-313
 - levels, list of, 37-38
 - performance tuning, 608-610
- Guidance Automation Extensions (GAX), 184
- Guidance Automation Toolkit (GAT), 184

H

- hardware accelerators, 701
- hardware encryption for performance tuning, 622
 - custom encryption, 623-625
 - message encryption, 623-624
 - transport encryption, 622-623
- HATEOAS (Hypermedia as the Engine of Application State), 323
- health and activity tracking tool (BizTalk Server), 194
- Hello World example. *See* Windows Azure, Hello World example
- high-availability ESB architecture, 480
 - distributed ESBs, 482-483
 - scaling, 481-482

- host container, communication with, 171-172
- host service, creating (Web service example), 233
- hosted cloud, 209
- hosting
 - co-hosting, 645-648
 - .NET Remoting components
 - in COM+ services, 57
 - in console applications, 57
 - in IIS, 57
 - in Windows services, 56
 - WCF services
 - IIS hosting, 113-114
 - managed Windows services, 112-113
 - REST service hosting, 115-116
 - selecting hosting environment, 108-110
 - self-hosted services, 110-112
 - Windows Activation Service (WAS), 114
- hosting environment in Windows
 - Server AppFabric, 188-189
- hosting models, selecting, 610-612
- hosts in BizTalk Server, 481
- hub-and-spoke integration model, 468
- hub-bus integration model, 469-470
- human intervention in business
 - processes, 450-451
- hybrid cloud authorization model (Access Control), 553-554
- hybrid cloud deployment model, 209
- hybrid clouds, 545
- Hypermedia as the Engine of Application State (HATEOAS), 323

I

- IaaS (Infrastructure-as-a-Service), 210-212
- ICCommunicationObject, closing. *See* cleaning up resources
- ICCommunicationObject.Abort()
 - method, 814
- ICCommunicationObject.Close()
 - method, 812-814
- Identity Metasystem, 533, 535
- identity providers, 537
 - developing, 542-543
- IDisposable interface, ClientBase and ChannelFactory classes, 815
- IDisposable.Dispose() method, 814-815
- IHttpHandler, REST service
 - processing, 74
- IIdentity interface, 522
- IIS (Internet Information Service)
 - hosting .NET Remoting components, 57
 - selecting hosting models, 612
- IIS hosting, 113-114
- imperative authorization, 524
- implementation requirements, service contracts, 693
- implicit programming model (System.Transactions library), 68-69
- implicit service discovery, 147-148
- in-line documentation, 322
- in-memory application cache in
 - Windows Server AppFabric, 190-191
- Increased Federation goal, 249-250
- Increased Intrinsic Interoperability
 - goal, 250, 285
- Increased Organizational Agility
 - goal, 285

- Increased ROI goal, 285
 - Increased Vendor Diversification
 - Options goal, 249-250
 - industry standards, list of, 70, 688-689
 - InfoCard, 536-540
 - infrastructure metrics, 654
 - Infrastructure-as-a-Service (IaaS), 210-212
 - input endpoints, Windows Azure roles, 221-222
 - instanting (WCF), 105-106
 - instantiating services (Hello World example), 226
 - integration models, 467-470
 - integrity, defined, 122
 - inter-organization service composition security, 545-546
 - Inter-Role Communication (IRC), 222-223
 - intercepting messages, 589
 - at caching utility service, 590
 - comparison of techniques, 591
 - at intermediary, 589
 - in service container, 589
 - at service proxy, 590
 - interceptors, 670
 - intercloud deployment model, 209
 - interface contracts
 - creating, 252
 - defined, 78
 - intermediary, intercepting messages, 589
 - Intermediate Routing design pattern, 133, 136, 748
 - internal cloud, 208
 - Internet Information Service (IIS), hosting .NET Remoting components, 57
 - interoperability, 40
 - interoperable bindings, 91
 - inventory analysis cycle, 337-338
 - inventory boundary design patterns. *See* Domain Inventory design pattern; Enterprise Inventory design pattern
 - Inventory Endpoint design pattern, 749
 - inventory standardization design patterns. *See* Canonical Protocol design pattern; Canonical Schema design pattern
 - inventory structure design patterns
 - Logic Centralization design pattern, 751
 - Service Layers design pattern, 169, 377-378, 779
 - Service Normalization design pattern, 781
 - Inversion of Control design pattern, 570
 - IPipelineContext interface (BizTalk BAM), 666
 - IPrincipal interface, 521
 - IRC (Inter-Role Communication), 222-223
 - itineraries
 - BizTalk Server, 201
 - ESB Toolkit, 472-474
 - lifecycle of*, 475-476
 - types of*, 474
- ## J
- Java RMI (Java Remote Invocation Call), 47
 - JIT resolution, 477
 - JSON encoding, configuring contracts for, 421-422
 - just-in-time activation (JITA), 50

K—L

large binary objects, transferring, 629
 Laws of Identity, 536
 layers (WCF), 149-150
 Legacy Wrapper design pattern, 750
 LINQ-to-XML mapping, 271-272
 listener adapters, 114
 locations in BizTalk Server
 messaging, 199
 logging messages (WCF), 153
 Logic Centralization design pattern, 751
 logic-to-contract coupling, 285
 logical models for metrics, defining, 661-662
 loose coupling. *See* Service Loose Coupling design principle
 Loose strategy (versioning), 102
 loosely couple events, 50

M

manageability extensions (Windows Server AppFabric), 192
 managed discovery, 141
 managed Windows services, 112-113
 management of BizTalk BAM
 database outages, 672
 reporting, 676
 scripting deployment, 673-676
 security, 672-673
 management tools (WCF)
 administration tools, 151
 message logging tools, 153
 troubleshooting tools, 151-152
 mappings
 LINQ-to-XML, 271-272
 Microsoft ESB to Enterprise Service Bus compound pattern, 490-492
 object-to-object, 269-271

memory resources, cleaning up, 812
 message buffers, Service Bus, 496-497
 creating, 514-516
 REST and, 499
 message contracts
 creating, 251
 defined, 79
 MessageContract attribute, 83-86
 message encryption, 623-624
 message filters, defining for routing services, 137
 message formats in .NET Remoting, 60
 message logging tools (WCF), 153
 message payload, 660
 message queues (MSMQ), 63-65
 Message Screening design pattern, 529, 752
 message security mode, 123
 message sizes (REST services), 621-622
 MessageContract attribute (WCF), 83-86
 messages
 intercepting, 589
 at caching utility service, 590
 comparison of techniques, 591
 at intermediary, 589
 in service container, 589
 at service proxy, 590
 itineraries. *See* itineraries
 receiving from Service Bus, 484-485
 sending to Service Bus, 485-486
 MSMQ, sending/receiving, 65-66
 messaging
 BizTalk Server, 193, 196
 pipelines, 197-198
 ports and locations, 199
 Service Broker (SSB), 160-164

Messaging Metadata design pattern, 84, 201, 251, 753
messaging property, 660
messaging-based middleware, 468
metadata
 BizTalk Server context properties, 200-201
 MEX endpoints, 100-101
Metadata Centralization design pattern, 147, 219, 325-326, 417, 754
metadata exchange (MEX), 101
metrics
 BAM. *See* BAM
 capturing data, 655-656
 logical model definition, 661-662
 service composition metrics, 669
 Standard Mold case study, 677-680
 types of, 654-655
 views
 creating, 663-664
 role-based views, 662
MEX (metadata exchange), 101
MEX endpoints, 100-101
microflow, 472
Microsoft, ESB and, 466-467
Microsoft Messaging Queue. *See* MSMQ
Microsoft Operations Manager (MOM), 654
mixed mode security, 124
Model-View-Controller design pattern, 568
Model-View-Presenter design pattern, 568
Model-View-View-Model design pattern, 568

Modern SOA Infrastructure, 6
modularity design patterns, 569-571
modules (Prism Library), 565-566
 in application lifecycle, 576
 dynamically loading, 579-581
MOM (Microsoft Operations Manager), 654
monitoring ESB, 488-489
MSMQ (Microsoft Messaging Queue), 63
 components of, 63
 message queues, 64-65
 messages, sending/receiving, 65-66
 service-orientation and, 66
MTOM encoding, 627-629
Multi-Channel Endpoint design pattern, 633, 755
multi-tenant cloud, 208
mutual authentication, 520, 536

N

n-tier architecture, 45-47
named pipes, 279
naming conventions
 service capabilities, 281-282
 services, 280
Native XML Web Services (SQL Server), 157-160
.NET assemblies, history of, 51
.NET Enterprise Services, 156
 history of
 COM+ services, 49, 51
 COM/DCOM, 48-49
 Distributed Transaction Coordinator (DTC), 52-53
 .NET assemblies, 51
 service-orientation and, 53

.NET Remoting, 47, 54

architecture of, 54, 56

configurations, 57

*activation types, 58, 60**communication protocols, 60**message formats, 60*

hosting components

*in COM+ services, 57**in console applications, 57**in IIS, 57**in Windows services, 56*

object lifetime management, 61

service-orientation and, 61-62

.NET System.Transactions library, 67

ambient transactions, 69

distributed resource transactions,
67-68explicit/implicit programming
models, 68-69**.NET types, generating, 258-264****Non-Agnostic Context design pattern,
169, 379-380, 756**

non-agnostic logic, defined, 32

non-repudiation, 552

notification service for this book series,
14, 810

notifications, 165, 670-671

numeric range dimension, 664

O**OAuth WRAP, 550**object lifetime management in .NET
Remoting, 61

object-to-object mapping, 269-271

Official Endpoint compound
pattern, 757

on-premise cloud, 208

On-ramp service (ESB Toolkit), 471

Open Authentication Web Resource
Access Protocol, 550operation attributes (WCF
transactions), 127-128

operation contracts, defined, 78

OperationContract attribute (WCF),
79-81optimization. *See* performance tuningorchestrated task service contracts in
BizTalk Server, 445-447orchestrated task services, 382-384,
423-425**orchestration**BizTalk Server and, 193, 388-391,
443-445*Compensating Service**Transaction design pattern,*
456-461*exception management,*
202-203*itineraries, 201**orchestrated task service*
*contracts, 445-447**Process Centralization design*
*pattern, 450-451, 454**State Repository design pattern,*
455-456*Superior Stamping case study,*
448-449*WS-* support, 447-448*

execution time, 385

optional design patterns, 388

versioning, 180

WF (Windows Workflow Foundation) and, 388-391, 395-397

- ASMX services, publishing workflows as*, 399-407
- Compensating Service Transaction design pattern*, 387, 434-435
- ExternalDataExchange services, publishing workflows via*, 413-416
- history of*, 397-398
- Process Abstraction design pattern*, 382-385
- Process Centralization design pattern*, 382-385, 425-426
- REST services, publishing workflows as*, 419-425
- Standard Mold case study*, 436-439
- State Repository design pattern*, 386, 426-433
- WCF 3.5 activities, publishing workflows via*, 408-410
- WCF 4.0 activities, publishing workflows via*, 410-413
- WS-I BasicProfile support*, 417-419

Orchestration compound pattern, 169, 194, 382, 395, 443, 758

orchestration schedule, 660

OrchestrationEventStream API (BizTalk BAM), 666

outbound-rendezvous relayed connectivity model (Service Bus), 494, 496

P

PaaS (Platform-as-a-Service), 211-212

page blobs in Windows Azure Storage, 243

parallelism, 641

- in BizTalk Server, 643-644
- iReplicatorActivity in WF, 644-645
- in WF, 641, 643

parameters

- passing to workflow instances, 178
- returning from workflow instances, 178-179

Partial State Deferral design pattern, 456, 759

Partial Validation design pattern, 760

partition keys in Windows Azure Storage, 240

Passive View design pattern, 568

patterns. See design patterns

performance policies, 612, 615-616, 621

performance tuning, 584

- optimization areas, 585-586
- service capabilities, 586
 - caching*, 587-591
 - caching implementation technologies*, 592-593
 - caching REST responses*, 599-601
 - coarse-grained service contracts*, 608-610
 - computing cache keys*, 593-594
 - hardware encryption*, 622-625
 - monitoring cache efficiency*, 601
 - MTOM encoding*, 627-629
 - performance policies*, 612, 615-616, 619-621

- reducing resource contention,*
603-604
 - request throttling,* 604-608
 - REST service message sizes,*
621-622
 - selecting hosting models,*
610-612
 - service contract design,* 630-632
 - service-orientation principles,*
 impact on, 633
 - Standard Mold case study,*
594-596
 - Superior Stamping case study,*
597-599
 - transports,* 625-627
- service compositions, 637, 648
 - asynchronous interactions,*
639-641
 - co-hosting,* 645-648
 - parallelism,* 641, 643-645
 - service-orientation principles,*
 impact on, 648
 - transformation avoidance,*
637-638
- state management and, 700-701
- persistence, workflow**
 - with AppFabric, 189-190
 - in WF, 170-171
- persistence service, 131**
 - in WF 3.0, 429-431
 - in WF 4.0, 431-433
- pipelines in BizTalk Server messaging,**
197-198
- Platform-as-a-Service (PaaS), 211-212**
- Plug-In design pattern, 570**
- point-to-point integration channels, 467**
- policies, effect on performance, 612,**
615-616, 619-621
- Policy Centralization design pattern, 761**
- PolicyActivity, 181**
- ports in BizTalk Server messaging, 199**
- Prentice Hall Service-Oriented***
 Computing Series from Thomas Erl,
 13-14, 810
- presentation layer**
 - design patterns, 567
 - modularity patterns,* 569-571
 - user interface patterns,* 567-569
 - Prism Library, 559
 - modules,* 565-566
 - regions,* 563-564
 - shared services,* 566
 - shell,* 561-562
 - views,* 562-563
 - service-oriented user interface
 - example, 571
 - dynamically loading modules,*
579-581
 - project creation,* 571-579
 - weaknesses in traditional models,
558-559
- Presentation Model design pattern,**
568-569
- Prism Library, 559**
 - modules,* 565-566
 - regions,* 563-564
 - service-oriented user interface
 - example, 571
 - dynamically loading modules,*
579-581
 - project creation,* 571-579
 - shared services,* 566
 - shell,* 561-562
 - views,* 562-563
- private assemblies, 51**
- private cloud deployment model, 208**

- private components in COM+, 50
- probe queries, 143-144
- Process Abstraction design pattern, 169, 378, 380, 382-385, 762
- process boundary (IIS), 113
- Process Centralization design pattern, 169, 382-385, 425, 450
 - human intervention in business processes, 450-451
 - profile, 763
 - workflow design in WF, 425-426
 - WS-BPEL support, 426, 451, 454
- production environment, tracking profiles in, 660
- productivity tools in BizTalk Server, 194
- programming model (WF), 176-177
- progress dimension, 664
- properties in Windows Azure Storage, 240-241
- Protocol Bridging design pattern, 136, 200, 274, 764
- protocol channels, 149
- protocols
 - Canonical Protocol design pattern, 274
 - component implementation, 278-279*
 - dual protocols, 279-280*
 - named pipes, 279*
 - REST services implementation, 277-278*
 - Web services implementation, 275-277*
 - cleaning up resources, 822
- Proxy Capability design pattern, 765
- proxy classes, writing, 118-119
- public cloud authorization model (Access Control), 554

- public cloud deployment model, 208
- publish-and-subscribe integration model, 468
- publishing WF workflows
 - as ASMX services, 399-407
 - via ExternalDataExchange services, 413-416
 - as REST services, 419-425
 - via WCF 3.5 activities, 408-410
 - via WCF 4.0 activities, 410-413
 - WS-I BasicProfile support, 417-419

Q—R

- Query Notification (SQL Server), 165
- queued components, 50
- queues
 - message queues (MSMQ), 63-65
 - in Windows Azure Storage, 239, 241-242
- rapid prototyping, 671
- real-time aggregations
 - dimensions in, 662
 - scheduled aggregations versus, 660-661
- Receive activity, configuring for REST services, 422-423
- receiving application (MSMQ), 63
- receiving messages
 - Service Bus, 484-485
 - MSMQ, 65-66
- recommended reading, 4-6, 41-42
- Redundant Implementation design pattern, 207, 213, 766
- regions (Prism Library), 563-564
- Reliable Messaging design pattern, 160, 767
- relying parties, 537, 541-542
- remotable classes in .NET Remoting, 56

Remote Method Invocation (RMI), 47
Remote Procedure Calls (RPC), 47
remoting. *See* .NET Remoting
ReplicatorActivity in WF, 644-645
reporting (BizTalk BAM management), 676
request throttling, 604-605
 with BizTalk Server, 607-608
 Superior Stamping case study, 606-607
 with WCF, 605-606
Resolver service (ESB Toolkit), 471
resolver strings for itineraries (ESB Toolkit), 475
resolvers (ESB Toolkit), 476-478
resource contention, reducing, 603-604
REST responses, caching, 599-601
REST services, 92-93
 Access Control (AppFabric) and, 552-553
 cleaning up resources, 822
 discoverability, 323
 dispatcher system in WCF, 77
 entity abstraction with (Standard Mold case study), 351-367
 hosting, 115-116
 message sizes, 621-622
 processing, 74
 publishing workflows as, 419-425
 resource naming conventions, 282
 Service Bus contracts, defining, 513-514
 Service Bus message buffers and, 499
 as service implementation option, 277-278
 services as, 31
 UriTemplate attribute, 96-98

 WCF-Custom adapter provider and, 479
 WebGet attribute, 93-95
 WebInvoke attribute, 95-96
 in Windows Azure, 235
 addressing, 235
 creating, 236-239
REST-based service consumers in Service Bus, 499
REST-based service design in Service Bus, 498
reusability. *See* Service Reusability design principle
RMI (Remote Method Invocation), 47
role-based authorization, 520-524
role-based security in COM+, 50
role-based views, 662
roles, 520
 Azure roles, 219
 input endpoints, 221-222
 Inter-Role Communication (IRC), 222-223
 virtual machines, 220-221
 Web roles, 220
 worker roles, 220
 composition roles, 377
 selecting (Hello World example), 224
router services (WCF Router), 132-133
 fault tolerance, 139-140
 routing configuration, 135-139
 routing contracts, 134-135
 RoutingService class, 133-134
routing contracts, 134-135
RoutingService class, 133-134
row keys in Windows Azure Storage, 241
RPC (Remote Procedure Calls), 47
Rules Centralization design pattern, 181, 388, 768

S**SaaS (Software-as-a-Service)**, 211**SAML**, 126**SAO (server-activated objects)**, 58**scalability**, 700**scaling ESB architecture**, 481-482**scaling out**, 608

in WF 3.0, 429-431

in WF 4.0, 431-433

scheduled aggregations, real-time

aggregations versus, 660-661

Schema Centralization design pattern,

217, 251, 253, 445, 769

schemas. *See* XML schemas**scripting deployment (BizTalk BAM**

management), 673-676

security

BizTalk BAM management,

672-673

in COM+, 50

Exception Shielding design pattern,

319-321

performance tuning, 612-621

Service Bus authentication, 508-509

terminology, 122

WCF security, 122-123

authorization, 125-126

brokered authentication, 519

claims-based authorization,

524, 526-529

direct authentication, 518-519

federated identity, 126

mutual authentication, 520

role-based authorization,

520-524

security modes, 123-125

Standard Mold case study, 530,

532-533

Windows Azure security, 543

Access Control (AppFabric),

548-554

cloud computing, 543-547

Standard Mold case study,

555-556

Windows Identity Foundation

(WIF), 533

Active Directory Federation

Services (ADFS), 539

digital identity, 534-536

identity providers, developing,

542-543

programming model, 540-541

relying parties, developing,

541-542

Windows Cardspace, 536-539

security modes, 123-125**security principal**, 521-522**Security Token Service (STS)**, 535, 546**security tokens**, 525, 534**self-hosted services**, 110-112**Send activity, configuring for REST**

services, 422-423

sending application (MSMQ), 63**sending messages**

MSMQ, 65-66

to Service Bus, 485-486

Separated Interface design pattern, 570**Separated Presentation design pattern**,

568-569

sequential workflows, 169**serializable classes in .NET**

Remoting, 56

serialization, 82-83**server-activated objects (SAO)**, 58**server-side itineraries**, 474

Service Abstraction design principle,
 26, 82, 84, 102, 104, 207, 217,
 313-314, 696
Service Agent design pattern, 133,
 589, 770
**service announcements, sending/
 receiving**, 144-146
service attributes (WCF transactions),
 129-130
**service authorization scenarios (Access
 Control)**, 553-554
service autonomy, 98
Service Autonomy design principle, 27,
 110, 190, 218, 699
Service Broker compound pattern, 136,
 160-164, 771
Service Bus, 494
 configuration, 504-512
 authentication, 508-509
 contract definition, 505-506
 contract implementation,
 506-507
 end-to-end security, 509
 service configuration, 508
 service host configuration,
 511-512
 service publication, 512
 as connectivity fabric, 494-496
 connectivity models, 499
 Eventing, 500
 Service Remoting, 501
 Tunneling, 501-502
 creating, 503
 message buffers, 496-497
 creating, 514-516
 REST and, 499
 receiving messages from, 484-485
 REST-based contracts, defining,
 513-514

REST-based service
 consumers, 499
 REST-based service design, 498
 sending messages to, 485-486
 service registry, 497-498
Service Callback design pattern, 772
service candidates, 35, 251
service capabilities
 naming conventions, 281-282
 performance tuning, 586
 caching, 587-591
 *caching implementation
 technologies*, 592-593
 caching REST responses,
 599-601
 coarse-grained service contracts,
 608-610
 computing cache keys, 593-594
 hardware encryption, 622-625
 monitoring cache efficiency, 601
 MTOM encoding, 627-629
 performance policies, 612,
 615-616, 619-621
 reducing resource contention,
 603-604
 request throttling, 604-608
 REST service message sizes,
 621-622
 selecting hosting models,
 610-612
 service contract design, 630-632
 service-orientation principles,
 impact on, 633
 Standard Mold case study,
 594-596
 Superior Stamping case study,
 597-599
 transports, 625-627

Service Composability design principle,
27, 125, 373, 560, 704-705

service composition

Capability Composition design
pattern, 374-375

Capability Recomposition design
pattern, 374-376

defined, 33-34

inter-organization service
composition security, 545-546

Non-Agnostic Context design
pattern, 379-380

performance tuning, 637, 648

asynchronous interactions,
639-641

co-hosting, 645-648

parallelism, 641, 643-645

service-orientation principles,
impact on, 648

transformation avoidance,
637-638

Process Abstraction design
pattern, 380

roles, 377

Service Composability design
principle, 373

Service Layers design pattern,
377-378

service-orientation and, 371-373

task services, 380

service composition metrics, 669

service consumers (WCF), 116-117

ChannelFactory class, 119-120

Service Metadata Tool, 117-118

writing proxy class, 118-119

**service container, intercepting
messages, 589**

service contracts, 704

defined, 36-37, 78

implementation example, 106-108

performance tuning, 630-632

ServiceContract and
OperationContract attributes,
79-81

versioning, 102-103

WF support, history of, 397-398

**Service Data Replication design
pattern, 190, 218, 773**

**Service Decomposition design
pattern, 774**

**service delivery models in cloud
computing, 210-212**

**Service Discoverability design
principle, 27, 147, 219, 321, 417, 535**

implementation requirements, 703

in-line documentation, 322

profile, 702-703

REST services, 323

service profiles, 323-324

**Service Encapsulation design pattern,
169, 332, 775**

service endpoints

attributes, 81

defined, 79

endpoint element, 86-87

address attribute, 88-89

binding attribute, 89-92

contract attribute, 92

**Service Façade design pattern, 304,
306-307, 776**

service framework processing, 586

Service Grid design pattern, 777

service granularity, defined, 37

- service hosting (WCF)**
 - IIS hosting, 113-114
 - managed Windows services, 112-113
 - REST service hosting, 115-116
 - selecting hosting environment, 108-110
 - self-hosted services, 110-112
 - Windows Activation Service (WAS), 114
- service implementation options, 274**
 - components, 278-279
 - REST services, 277-278
 - Web services, 275-277
- service implementation processing, 585**
- Service Instance Routing design pattern, 778**
- service inventory, defined, 34**
- service inventory blueprints, 34**
- Service Layers design pattern, 169, 377-378, 779**
- service level agreement (SLA)**
 - enforcement, 488
- Service Locator design pattern, 571**
- Service Loose Coupling design principle, 26, 81-82, 84, 217, 285-286, 559**
 - design patterns and, 286
 - profile, 695
 - service capability granularity and, 308-313
- Service Messaging design pattern, 84, 780**
- Service Metadata Tool, 117-118**
- service metrics, 654**
- service model layer, 149**
- service modeling, 251**
- service models, defined, 31-32**
- service namespace, 503, 551**
- Service Normalization design pattern, 781**
- service package (Web service example)**
 - creating and deploying, 233
 - promoting to production, 234
- Service Perimeter Guard design pattern, 529, 782**
- service profiles, discoverability, 323-324**
- service proxy, intercepting messages, 590**
- Service Refactoring design pattern, 217, 783**
- service registry (Service Bus), 497-498**
- Service Remoting connectivity model (Service Bus), 501**
- Service Reusability design principle, 26, 125, 329-330, 337, 339, 697-698, 704**
- Service Statelessness design principle, 27, 171, 190, 700-701**
- service-orientation**
 - defined, 25-27
 - history of .NET Enterprise Services and, 53
 - MSMQ and, 66
 - .NET Remoting and, 61-62
 - service composition and, 371-373
- service-orientation principles, impact of performance tuning, 633, 648**
- service-oriented analysis, defined, 34-35**
- Service-Oriented Architecture: Concepts, Technology, and Design, 5***
- service-oriented architecture. *See* SOA**
- service-oriented computing**
 - defined, 25
 - goals of, 40-41

service-oriented design, defined, 35-36

service-oriented user interface

example, 571

dynamically loading modules,
579-581

project creation, 571-579

**service-related granularity, defined,
37-38**

service-types, 81

**ServiceContract attribute (WCF),
79-81**

services

as components, 29

defined, 28-29

instantiating (Hello World
example), 226

naming conventions, 280

protection patterns, 529

as REST services, 31

scalability, 700

WCF. *See* WCF

as Web services, 30

shared assemblies, 51

shared services (Prism Library), 566

shell (Prism Library), 561-562

silo-based applications, 467

**Silverlight, 560. *See also* service-
oriented user interface example**

SingleCall objects, 58

Singleton objects, 59

SLA enforcement, 488

**SOA (service-oriented architecture)
defined, 27
explained, 47**

**SOA Certified Professional (SOACP),
14. *See also* www.soaschool.com**

SOA Design Patterns, 5

SOA Governance, 6

SOA Magazine, The Web site, 14, 810

SOA Manifesto, 27

annotated version, 796-808

original version, 796

SOA Principles of Service Design, 5, 11

SOA with Java, 6

SOA with REST, 6

**SOACP (SOA Certified Professional),
14. *See also* www.soaschool.com**

SOAP

attachments, 701

message contracts, 83-86

processors, 701

SOAP Faults, 99

SOAP service in COM+, 50

Software Factories, 184

Guidance Automation Extensions
(GAX), 184

Guidance Automation Toolkit
(GAT), 184

Web Services Software Factory,
184-186

Software-as-a-Service (SaaS), 211

Software-plus-Services, 206

specifications, 70, 688-689

SQL Azure, 217-218

SQL Express configuration, 224

SQL persistence services

in WF 3.0, 429-431

in WF 4.0, 431-433

SQL Server, 156-157

Query Notification, 165

Service Broker, 160-164

Web services support, 157-160

XML support, 165-166

SQL Server Service Broker Adapter, 200

**Standard Mold case study. *See* case
studies; code examples**

Standardized Service Contract design
 principle, 26, 78, 81-82, 84, 217, 250, 274, 340, 399
 contract-first development, 250-252
 design patterns and, 252
 profile, 693-694
standards, 70, 688-689
state machine workflows, 169
state management, 700-701
State Messaging design pattern, 456, 784
State Repository design pattern, 171, 207, 386, 426-428
 BizTalk Server, 455-456
 persistence service and scaling out
 in WF 3.0, 429-431
 persistence service and scaling out
 in WF 4.0, 431-433
 profile, 785
Stateful Services design pattern, 190, 207, 213, 456, 590, 786
storage services
 creating (Web service example), 233
 in Windows Azure, 239-240
 blobs, 242-243
 queues, 241-242
 tables, 240-241
 Windows Azure Drive, 243
Strict strategy (versioning), 102
STS (Security Token Service), 535, 546
subjects. See clients
Superior Stamping case study. See case studies; code examples
Supervising design pattern, 568
Supervising Presenter design pattern, 569
svcutil utility program, 258
 code generation with, 294-297

symbols
 color in, 13
 legend, 13
synchronization, 50
system-provided bindings, 90
System.Transactions library, 67
 ambient transactions, 69
 distributed resource transactions, 67-68
 programming models, explicit/implicit, 68-69
Systems Center Operations Manager Management Pack for BizTalk Server, 481

T

tables in Windows Azure Storage, 239-241
target endpoints, defining for routing services, 136-137
task service model, defined, 31
task services, 380. See also orchestrated task services
Termination Notification design pattern, 102, 787
Three-Layer Inventory compound pattern, 378, 788
throttling. See request throttling
throttling controls, 105
time dimension, 664
timestamps in Windows Azure Storage, 241
tokens, 546, 551
TPE (Tracking Profile Editor), 659-660
tracing (WCF), 151-152
tracking host, 672
Tracking Profile Editor (TPE), 659-660

tracking profiles, 659-660

TransactionAutoComplete
attribute, 128

TransactionAutoCompleteOnSession-
Close attribute, 130

TransactionFlow attribute, 128

TransactionIsolationLevel
attribute, 129

transactions, 50. *See also*

System.Transactions library

ambient transactions, 69

distributed resource transactions,
67-68

explained, 51

in Windows Azure Storage, 241

WCF, 127

durable services, 131-132

operation attributes, 127-128

service attributes, 129-130

TransactionScopeRequired
attribute, 128

TransactionTimeout attribute, 130

Transformation service
(ESB Toolkit), 471

transformations. *See also* Data Model

Transformation design pattern

avoiding, 637-638

XSLT transformations, 272-273

transition plans for ESB, 489

transport channels, 149

transport encryption, 622-623

transport security mode, 123

transports, performance tuning,
625-627

troubleshooting management tools
(WCF), 151-152

trust, 546

trust boundaries, 122

Trusted Subsystem design pattern, 789

try-catch-finally-abort block, cleaning
up resources, 817

try-close-catch-abort block, cleaning up
resources, 818-819

tuning. *See* performance tuning

Tunneling connectivity model (Service
Bus), 501-502

U

UDDI, 325-326

UI Mediator design pattern, 568, 790

Uniform Contract design pattern, 282

Uniform Resource Identifiers (URIs),
explained, 88

UriTemplate attribute (REST services),
96-98

user interface design patterns, 567-569

user interface example. *See* service-
oriented user interface example

using block, cleaning up resources, 816

Utility Abstraction design pattern, 169,
182, 335, 377-378

profile, 791

with .NET Web service (Standard
Mold case study), 339-351

utility logic (Application Blocks),
182-183

utility methods, cleaning up resources,
819, 821

utility service model, defined, 32

V

Validation Abstraction design pattern,
82, 217, 315-318, 792

vendor diversification, 40

version control systems, 698

Version Identification design pattern,
102, 793

versioning

- orchestrations, 180
- service contracts, 102-103

view discovery, view injection

versus, 563

view injection, view discovery

versus, 563

views

- creating, 663-664
- Prism Library, 562-563
- role-based views, 662

virtual machines, Windows Azure roles, 220-221**virtual private cloud, 209****Visual Studio**

- creating schemas, 254, 256-258
- elevated mode, 223

W**WAS (Windows Activation Service), 114, 612****WATs (Windows Azure Tables), 240-241****WCF (Windows Communication Foundation)**

- administration management tools, 151
- behaviors, 104-105
- cleaning up resources, 811-812
 - Abort() method (ICommunicationObject), 814*
 - based on bindings, 822*
 - ClientBase and ChannelFactory classes, 815*
 - Close() method (ICommunicationObject), 812-814*

Dispose() method

(IDisposable), 814-815

try-catch-finally-abort block, 817

try-close-catch-abort block, 818-819

using block, 816

utility methods, 819, 821

data contracts

DataContract attribute, 82-83

defined, 78

fault contracts, 98-100

extensibility

channel bindings, 148

channel layer extensibility, 150

layers, 149-150

instancings, 105-106**interceptors, 670****interface contracts, defined, 78****message contracts**

defined, 79

MessageContract attribute, 83-86

message logging tools, 153**MEX endpoints, 100-101****operation contracts, defined, 78****overview, 76-77****programming model for WF**

orchestrations, 397

request throttling, 605-606**REST services, 92-93**

UriTemplate attribute, 96-98

WebGet attribute, 93-95

WebInvoke attribute, 95-96

security, 122-123

authorization, 125-126

brokered authentication, 519

claims-based authorization, 524, 526-529

- direct authentication*, 518-519
- federated identity*, 126
- mutual authentication*, 520
- role-based authorization*, 520-524
- security modes*, 123-125
- Standard Mold case study*, 530, 532-533
- service consumers, 116-117
 - ChannelFactory class*, 119-120
 - Service Metadata Tool*, 117-118
 - writing proxy class*, 118-119
- service contracts
 - defined*, 78
 - implementation example*, 106-108
 - ServiceContract and OperationContract attributes*, 79-81
 - versioning*, 102-103
- service endpoints
 - address attribute*, 88-89
 - binding attribute*, 89-92
 - contract attribute*, 92
 - defined*, 79
 - endpoint element*, 86-87
- service hosting
 - IIS hosting*, 113-114
 - managed Windows services*, 112-113
 - REST service hosting*, 115-116
 - selecting hosting environment*, 108-110
 - self-hosted services*, 110-112
 - Windows Activation Service (WAS)*, 114
- terminology, 78-79
- transactions, 127
 - durable services*, 131-132
 - operation attributes*, 127-128
 - service attributes*, 129-130
- troubleshooting management tools, 151-152
- WCF 3.5 activities, publishing workflows via**, 408-410
- WCF 4.0 activities, publishing workflows via**, 410-413
- WCF Adapter**, 200
- WCF Discovery**, 140-141
 - discovery modes*, 141-143
 - Discovery Proxies*, 146
 - implicit service discovery*, 147-148
 - probe queries*, 143-144
 - sending/receiving service announcements*, 144-146
- WCF Router**, 132-133
 - fault tolerance*, 139-140
 - routing configuration*, 135-139
 - routing contracts*, 134-135
 - RoutingService class*, 133-134
- WCF-Custom adapter provider, REST services and**, 479
- WCF-to-WCF bindings**, 91
- Web roles (Windows Azure)**, 220
- Web Service Contract Design and Versioning for SOA**, 5, 294
- Web Service Enhancements (WSE)**, 73
- Web service example. See Windows Azure, Web service example**
- Web services. See also services**
 - ASMX (XML Web services)*, 71, 73
 - as service implementation option*, 275-277
 - services as*, 30

- SQL Server support for, 157-160
- WSE (Web Service Enhancements), 73
- Web Services Adapter, 199**
- Web Services Software Factory, 184-186**
- Web sites**
 - www.serviceorientation.com, 810
 - www.sliverlight.net, 560
 - www.soa-manifesto.com, 5, 796
 - www.soa-manifesto.org, 27, 796
 - www.soabooks.com, 6, 13-14, 42, 810
 - www.soaglossary.com, 5, 13, 24, 42, 810
 - www.soamag.com, 14, 810
 - www.soapatterns.org, 5, 810
 - www.soaprinciples.com, 5, 42, 810
 - www.soaschool.com, 14
 - www.soaspecs.com, 5, 13, 70, 141, 560, 688, 810
 - www.whatissoa.com, 5, 41, 810
- Web-oriented architecture (WOA), 207**
- WebGet attribute (REST services), 93-95**
- WebInvoke attribute (REST services), 95-96**
- WebMethod attribute, 72**
- WebService attribute, 71-72**
- WF (Windows Workflow Foundation), 76, 166**
 - activities, 172-175
 - architecture, 167-168
 - business rules, 180-181
 - extensibility, 180
 - host container communication, 171-172
 - interceptors, 670
 - orchestration and, 388-391, 395-397
 - ASMX services, publishing workflows as, 399-407*
 - Compensating Service Transaction design pattern, 434-435*
 - ExternalDataExchange services, publishing workflows via, 413-416*
 - history of, 397-398*
 - Process Abstraction design pattern, 382-385*
 - Process Centralization design pattern, 382-385, 425-426*
 - REST services, publishing workflows as, 419-425*
 - Standard Mold case study, 436-439*
 - State Repository design pattern, 386, 426-433*
 - WCF 3.5 activities, publishing workflows via, 408-410*
 - WCF 4.0 activities, publishing workflows via, 410-413*
 - WS-I BasicProfile support, 417-419*
 - parallelism in, 641, 643-645
 - passing parameters to workflow instances, 178
 - programming model, 176-177
 - returning parameters from workflow instances, 178-179
 - versioning orchestrations, 180
 - workflow designer, 169-170
 - workflow persistence, 170-171
 - workflow runtime environment, 175
 - workflow types, 168-169
 - workflow-enabled services, 179

WF 3.0, persistence service and scaling out, 429-431

WF 4.0, persistence service and scaling out, 431-433

WIF (Windows Identity Foundation), 533

Active Directory Federation

Services (ADFS), 539

digital identity, 534-536

identity providers, developing, 542-543

programming model, 540-541

relying parties, developing, 541-542

Windows Cardspace, 536-539

WIF (Windows Identity Framework), 76

Windows Activation Service (WAS), 114, 612

Windows Azure. *See also* cloud computing

access control in, 528

application container, 216-217

cloud deployment models and, 210

cloud-based services, categories of, 215-216

ESB and, 483

receiving messages from Service Bus, 484-485

sending messages to Service Bus, 485-486

Hello World example, 223

cloud service project,

creating, 224

roles, selecting, 224

service, instantiating, 226

solution, creating, 225

platform overview, 213-216

REST services, 235

addressing, 235

creating, 236-239

roles, 219

input endpoints, 221-222

Inter-Role Communication (IRC), 222-223

virtual machines, 220-221

Web roles, 220

worker roles, 220

SQL Azure, 217-218

storage services, 239-240

blobs, 242-243

queues, 241-242

tables, 240-241

Windows Azure Drive, 243

Web service example, 227-232

Diagnostic Monitor, 229

host service and storage service, creating, 233

IOrderService interface

contract, 231

logging and diagnostic

APIs, 229

Order data contract, 231

output of IOrderService

interface contract and Order data contract, 232

service package, creating and deploying, 233

service package, promoting to production, 234

ServiceConfiguration.cscfg, 228

ServiceDefinition.csdef, 228

tracing listener creation, 230

WebRole.cs, 227

Windows Azure Drive in Windows Azure Storage, 239, 243

Windows Azure Fabric Controller, 214

Windows Azure platform AppFabric, 218. *See also* Access Control; Service Bus

Windows Azure security, 543

Access Control (AppFabric), 548-553
 service authorization scenarios, 553-554
 Standard Mold case study, 555-556

cloud computing, 543-547

Windows Azure Tables (WATs), 240-241

Windows Cardspace, 76, 536-541

Windows Communication Foundation.

See WCF

Windows Identity Foundation (WIF), 533

Active Directory Federation Services (ADFS), 539
 digital identity, 534-536
 identity providers, developing, 542-543
 programming model, 540-541
 relying parties, developing, 541-542
 Windows Cardspace, 536-539

Windows Identity Framework (WIF), 76

Windows Management

Instrumentation (WMI), 151

Windows Presentation Foundation.

See WPF

Windows Server AppFabric, 187

configurable hosting environment, 188-189
 Event Collector Service, 192

in-memory application cache, 190-191
 manageability extensions, 192
 service namespaces, creating, 503
 workflow persistence, 189-190

Windows services, hosting .NET

Remoting components, 56

Windows Workflow Foundation. *See* WF

wire transmission processing, 586

WMI (Windows Management

Instrumentation), 151

WOA (Web-oriented architecture), 207

worker roles (Windows Azure), 220

workflow design, 425-426

workflow designer (in WF), 169-170

workflow instances

passing parameters to, 178

returning parameters from, 178-179

workflow persistence

AppFabric, 189-190

WF, 170-171

workflow runtime environment

(in WF), 175

workflow-enabled services, 179

workflows. *See also* WF (Windows Workflow Foundation)

history of, 397-398

publishing

as ASMX services, 399-407

via ExternalDataExchange services, 413-416

as REST services, 419-425

via WCF 3.5 activities, 408-410

via WCF 4.0 activities, 410-413

WS-I BasicProfile support,

417-419

types of, 168-169

WPF (Windows Presentation Foundation), 76

- Prism Library, 559
 - modules*, 565-566
 - regions*, 563-564
 - shared services*, 566
 - shell*, 561-562
 - views*, 562-563

WS-AtomicTransaction, 127, 458

WS-BPEL, 385, 426, 451, 454

WS-Coordination, 127

WS-Discovery, 140

WS-I Basic Security Profile, 125

WS-I Basic Profile, 417-419

WS-MetadataExchange, 101, 417, 536

WS-Policy, 694

WS-SecureConversation, 125

WS-Security, 125, 536

WS-SecurityPolicy, 125, 536

WS-Trust, 125

WSCF.blue, code generation with,
297-301

WSCF.classic, code generation with,
302-304

WSDL, 77, 694

WSDL-first design approach, 289-294

- svcutil* utility program, code
generation with, 294-297
- WSCF.blue*, code generation with,
297-301
- WSCF.classic*, code generation
with, 302-304

WSE (Web Service Enhancements),
70, 73

X—Z

XAML (Extensible Application Markup Language), 560

XML parsers, 701

XML support in SQL Server, 165-166

XML Schema Definition Language, 82,
253, 694

XML schemas

- creating, 254, 256-258
- WSDL-first design approach,
289-294
 - svcutil* utility program, code
generation with, 294-297
 - WSCF.blue*, code generation
with, 297-301
 - WSCF.classic*, code generation
with, 302-304

XML serializers, 82-83

XML Web services (ASMX), 71, 73

XML-Encryption, 125

XML-Signature, 125

xsd.exe utility program, 264

XSLT, 638, 272-273