ADOBE COLDFUSION 10
ColdFusion 10 Enhancements and Improvements

web application construction kit

Ben Forta
with Charlie Arehart, Rob Brooks-Bilson, Raymond Camden, Kenneth Fricklas, Hemant Khandelwal, and Chandan Kumar
Dedications

Charlie Arehart
I’d like to dedicate this volume with thanks to the hundreds of speakers who’ve presented on the Online ColdFusion Meetup (coldfusion-metup.com), stepping up to the plate to share their knowledge and experience in all things ColdFusion.

Raymond Camden
To my wife, Jeanne. I love you.

Kenneth Fricklas
I dedicate this book to my wife and son, whom I told I wouldn’t do this again, but here we are.

Hemant Khandelwal
To my wife, Meera, and my two kids, Tishya and Tarush, who all have brought tremendous joy into my life.

Chandan Kumar
To mom with love.
AUTHOR BIOGRAPHIES

**Ben Forta** has more than two decades of experience in the computer industry, in product development, support, training, and marketing. As Adobe's director of developer relations, he is responsible for the company’s technical evangelism, community relations, and developer education programs and is a primary liaison between the company and the Adobe developer community. Ben is the author of more than 40 books, including best-selling titles on SQL and ColdFusion and titles on Microsoft Windows development, Regular Expressions, and Java. Over half a million Ben Forta books have been printed in English, and titles have been translated into 15 languages. Ben continues to write and blog at http://forta.com/ and present on web and application development topics worldwide. You can also find him on Twitter at @benforta.

A veteran ColdFusion developer and troubleshooter since 1997, with more than three decades in enterprise IT, **Charlie Arehart** is a longtime contributor to the ColdFusion community and has for several years been a recognized Adobe Community Professional, Adobe Forums MVP, ColdFusion Customer Advisory Board member, and more. An independent consultant, he provides short-term, remote, on-demand troubleshooting and tuning assistance for organizations of all sizes and ColdFusion experience levels (carehart.org/consulting). Besides running the 2800-member Online ColdFusion Meetup (coldfusionmeetup.com, an online ColdFusion user group), he hosts the UGTV repository for recorded presentations from hundreds of speakers (carehart.org/ugtv), the CF411 site offering more than 1800 tools and resources for ColdFusion users (cf411.com), and the CF911 site offering troubleshooting resources (cf911.com). A Certified Advanced ColdFusion developer and instructor for each version since ColdFusion 4, Charlie has spoken at nearly all the ColdFusion conferences worldwide and was a contributor to all three volumes of the ColdFusion 8 and 9 ColdFusion Web Application Construction Kit books.

**Rob Brooks-Bilson** is a consultant and author and the director of architecture and application development at Amkor Technology. He’s a frequent speaker at industry conferences and at local user groups. He is also the author of two O’Reilly books: *Programming ColdFusion* and *Programming ColdFusion MX*. Outside work, Rob is a technophile, blogger, photographer, bed jumper, world traveler, hiker, mountain biker, and Adobe Community Professional for ColdFusion. You can subscribe to Rob’s blog at rob.brooks-bilson.com and follow him on Twitter at @styggiti.

**Raymond Camden** is a senior developer evangelist for Adobe. His work focuses on web standards, mobile development, and ColdFusion. He is a published author and presents at conferences and user groups on a variety of topics. Raymond can be reached at his blog (www.raymondcamden.com), at @cfjedimaster on Twitter, or by email at raymondcamden@gmail.com.

**Kenneth Fricklas** has been using ColdFusion since Version 1.5 and teaching it since Version 3.0. A well-known speaker and author, he is currently the vice president of software engineering at Placewise Media in Denver, Colorado.
**Hemant Khandelwal** manages the ColdFusion products at Adobe and has built and shipped ColdFusion Server 8, 9, and 10 and ColdFusion Builder 1 and 2. He has several years of R&D experience in application-server internal design and Internet architecture and wrote the world’s first EJB2.0 container. He was part of the expert group committee for the J2EE 1.4, EJB2.0, and EJB3.0 specifications. He is a regular speaker at conferences and is passionate about ColdFusion and the role it plays in making hard things easy. He can be reached on Twitter at @khandelwalh.

**Chandan Kumar** has been part of the core development team at Adobe for ColdFusion for almost seven years and has been involved in managing its language and runtime application and keeping it current with the latest trends, including closures, REST support, caching upgrades, and PDF features. He has been a speaker at several conferences throughout the world, including Adobe MAX, SOTR, WebDU, CFUnited, and CFUG conferences and many e-seminars. He is a graduate of the Indian Institute of Technology.
ACKNOWLEDGMENTS

Ben Forta: Thanks to my co-authors, Charlie Arehart, Rob Brooks-Bilson, Raymond Camden, Kenneth Fricklas, Hemant Khandelwal, and Chandan Kumar, for their outstanding contributions. Although this book is affectionately known to thousands as “the Ben Forta book,” it is, in truth, as much theirs as it is mine. An extra thank you to Hemant Khandelwal for his tech review and for stepping up and taking on extra work when we needed additional contributing authors. Thanks to Nancy Peterson and Judy Ziajka for so ably shepherding this book through the publication process yet again. Thanks to the thousands of you who write to me with comments, suggestions, and criticism (thankfully not too much of the latter)—I do read each and every message (and even attempt to reply to them all, eventually), and all are appreciated. And last, but by no means least, a loving thank you to my wife, Marcy, and our children for putting up with (and allowing) my often hectic work schedule. Their love and support make all that I do possible.

Charlie Arehart: Where would we all be, in terms of our ColdFusion experiences, without Ben?! Most of us learned ColdFusion from him and/or this series, or from the official ColdFusion training that he helped shape, or from his blog, etc. He’s been the rock of the ColdFusion world from the beginning. (And he’s been a rock star, too. Those of us who saw him roll out on stage in the Hummer at “Powered by Detroit” in 2005 will never forget that scene!) And I’ve had the sincere pleasure and honor of contributing to the past seven volumes in this series and can attest to his being the glue that holds it all together. Thanks also to my fellow authors, and indeed to all in the community and at Adobe, who’ve contributed knowledge and experience that has helped all of us learn and improve over the years and even decades. This book is just one more reflection of the ColdFusion community’s continued, valuable support for one another. Finally, you see most writers thank their spouses, and it’s not just a formality, as they lose us for long periods of time during such projects, especially when we write on top of doing our other work. Kim, as I say daily for some reason or another, “Thank you, my love.” I thank God for bringing you into my life, and indeed for all I have and am.

Raymond Camden: I’d like to thank Ben for inviting me back to the book again and to thank the ColdFusion team for their help and support in writing this book.

Hemant Khandelwal: I would like to thank Ben for giving me the opportunity to be part of this book. A special thanks to Judy Ziajka for her help during chapter reviews and to my teammates at the ColdFusion product team who are the real heroes in building these features. I also want to thank my mom for what she has taught me. And finally, I want to thank my wife for her love.

Chandan Kumar: First, I would like to thank Ben Forta and Peachpit for believing in me and giving me this great opportunity to write. Then I would like to thank my manager, Hemant Khandelwal, for his continuous help and guidance throughout my career. I also want to thank my family and friends for their love and support and, finally, my lovely wife, Sudha Singh, for standing by me all along.
# CONTENTS AT A GLANCE

## PART I  Web Technology Innovation 1
- CHAPTER 1  ColdFusion 10 and HTML5 3
- CHAPTER 2  Using WebSocket 9
- CHAPTER 3  Charting Revisited 35
- CHAPTER 4  Web Services 55
- CHAPTER 5  Using REST Web Services 77
- CHAPTER 6  Embedding Video 93

## PART II  Rapid Development 105
- CHAPTER 7  CFML Enhancements 107
- CHAPTER 8  CFScript Enhancements 127
- CHAPTER 9  Object Relational Mapping Enhancements 149
- CHAPTER 10  Enhanced Java Integration 165
- CHAPTER 11  XML Enhancements 187

## PART III  Enterprise Ready 197
- CHAPTER 12  ColdFusion in the Cloud 199
- CHAPTER 13  Improved Administration 217
- CHAPTER 14  Scheduling 225
- CHAPTER 15  Security Enhancements 251
- CHAPTER 16  Improving Performance 265
- CHAPTER 17  Improved Integration 281
- CHAPTER 18  Apache Solr 297
- CHAPTER 19  Miscellaneous Enhancements 309

*Index* 325
This page intentionally left blank
PART I  Web Technology Innovation  

CHAPTER 1  ColdFusion 10 and HTML5

What’s New in HTML5?  
  Can I Use the New HTML5 Features?  
  ColdFusion 10 HTML 5 Support  
Geolocation and the <cfmap> Tag
  Getting a Google Maps API Key  

CHAPTER 2  Using WebSocket

What Exactly Is WebSocket?
  What About Browser Support?
Getting Started
Using the JavaScript API
Working with CFC Handlers
Using Server-Side Functions
Filtering Messages
  Working with Subchannels
  Using Selectors
Providing Security
  Using onWSAuthenticate
  Using Single Sign-On Mode
Generating Messages with CFCs
  Point-to-Point WebSocket
  Error Handling and Unsupported Browsers
ColdFusion Administrator Options

CHAPTER 3  Charting Revisited

Charting and ColdFusion 10
  Getting Started with <cfchart>
  Using Query Data in Charts
  Previewing and Zooming
ColdFusion 10 Chart Types
Styling a Chart
  Rendering Options and Fallback
  Plotting Multiple Series in a Chart
  Using Clickable URLs and JavaScript
  Highlighting Important Data
Refreshing Data and Using Live Data
Conclusion
Skinning Player Control  
Callback Events and Error Logging  
Other Enhancements  
  Loop Playback Support  
  Poster Image Support  
  Video Title Support  
  Playlist Support  
Extending the Media Player  
  Playing YouTube Video  
  Streaming  
  Digital Rights Management

PART II  Rapid Development

CHAPTER 7  CFML Enhancements
  Working with Closures  
    Arrays and Inline Functions  
    Lists and Inline Functions  
    Structs and Inline Functions  
  Miscellaneous Language Enhancements  
    <cfinclude> Gets More Intelligent  
    Application.cfc and Abort Handling  
    Handling Form Fields with the Same Name  
    ColdFusion Component Updates  
    Improvements to queryNew and queryAddRow

CHAPTER 8  CFScript Enhancements
  Evolving Script-Based Development in CFML
  Support for Colon Separators in Structures (Key:Value Pairs)
  Enhanced Script Looping (for-in over Queries and Java Arrays)
    Using for-in Looping over Queries
    Using for-in for Java Arrays
  Two New Tag Equivalents as Script Statements (setting and cookie)
    Using the setting Statement (for CFSETTING)
    Using the cookie Statement (for CFCookie)
  One New Tag Equivalent as a Function (invoke)
    Using the invoke Function (for CFInvoke)
    Invoking a Web Service Method
  Three New Tag Equivalents as CFCs (collection, index, and search)
    Using the collection CFC (for CFCollection)
    Using the index CFC (for CFIndex)
    Using the search CFC (for CFSearch)
  About These Adobe-Provided CFCs
CHAPTER 9  Object Relational Mapping Enhancements  149
ORM Enhancements Summary  149
ORM Search Enhancements  149
   Enabling ORM Entities for Indexing  151
   Indexing an ORM Entity  152
   Searching an ORM Entity  152
   Searching Multiple Entity Types  154
   Performing Offline Searches  155
   Searching Relationships  157
Managing Indexes  157
   Explicit Indexing  157
   Implicit Indexing  158
   Purging  158
Specifying Search Settings  158
   Server Level  158
   Application Level  159
   Component Level  159
   Property Level  160
HQL Logging Enhancements  162
Improved Handling of the ORM Hierarchy  163
CHAPTER 10  Enhanced Java Integration  165
About CFML and Java Integration  165
   Simple Java Example: DNS Lookup  166
   Using Java Libraries and Classes  167
   More About CFML and Java Integration  167
New: Dynamic Application-Specific Class Loading with this.javaSettings  168
   About Java Class Loading in ColdFusion  168
   Defining an Application-Specific Class Library  169
   Example: Naming a Load Path  170
   Example: Using a Better Reverse-DNS Java Library  171
   Reloading Changed Classes: reloadOnChange, watchInterval, and watchExtensions Arguments  172
   Challenges Using Dynamic Class Loading  173
Enhancement: Calling CFCs from Java with CFCProxy  175
   About the Long-Existent and Little-Known CFCProxy  176
   About the New directInvoke Argument  181
New: Proxying CFCs as Java Objects with createDynamicProxy  182
   Step 1: Define an Interface  182
   Step 2: Create a CFC That Offers an Implementation  183
   Step 3: Create a Java Class to Work with the CFC  183
   Step 4: Create a CFML Page to Pass the CFC to the Java Class  184
New: Looping over Java Arrays in CFML  185
Conclusion  186
## CONTENTS

**CHAPTER 11 XML Enhancements** 187
- ColdFusion XML and XPath 187
  - XPath and Variables 192
- ColdFusion XML and XSLT 192

**PART III Enterprise Ready** 197

**CHAPTER 12 ColdFusion in the Cloud** 199
- What Is “The Cloud”? 199
  - Cloud Technology 200
  - Advantages of the Cloud 201
  - Cloud Service Models 201
  - Cloud Deployment Models 203
- Amazon Web Services 203
  - Registering with AWS 204
  - Amazon Elastic Compute Cloud 204
  - Amazon Simple Storage Service 207
  - Other Notable AWS Components 210
- Designing for the Cloud 210
  - Main Differences Between Cloud and Traditional Designs 210
  - Design Considerations 212
- ColdFusion AMI 215

**CHAPTER 13 Improved Administration** 217
- What’s Not Covered 217
- The New Look: Shiny! 217
  - Caching Updates 219
- Security Updates 219
  - Automatic Logging of Administration Changes 219
  - Password Reset Script 220
  - Enable RDS (After Installation) 220
  - Restrict Admin Access by IP Address 221
- Server Updates 221

**CHAPTER 14 Scheduling** 225
- A Word About ColdFusion Standard versus Enterprise 225
- Introducing the Quartz Scheduling Engine 226
- Changes to CF_SCHEDULE 226
  - Setting the Stage 226
  - Application-Level Tasks 226
  - Grouping 228
  - Deleting Tasks 230
  - Running Tasks 230
  - Task Prioritization 230
  - Chaining 231
  - Excluding Dates 233
Repeating Tasks 234
Cron Expressions 235
Handling Exceptions 237
Handling Misfires 238
Event Handling 239
Listing Scheduled Tasks 243
Pausing and Resuming Tasks 246
Changes to the ColdFusion Administrator 247
Adding and Editing Tasks 247
Clustering Scheduled Tasks 248
Customizing the Quartz Scheduler 250

CHAPTER 15 Security Enhancements 251
ColdFusion Security Options 251
The Secure Profile 252
XSS Protection 253
CSRF Protection 257
File Upload Protection 260
Session Improvements 261
Miscellaneous Improvements 262

CHAPTER 16 Improving Performance 265
What Is Caching? 265
Caching Enhancements 266
Application-Specific Caching 266
Cache Regions 267
Changes to Built-in Functions 269
New Cache Functions 271
Enhanced Query Caching 273
Ehcache Version Update 275
Server Monitor Support for Caching 275
Other Performance Improvements: JVM Tuning 277

CHAPTER 17 Improved Integration 281
Improved Integration with Microsoft Exchange Server 281
Microsoft Exchange Server 2010 Support 281
Folder Operations Support 283
  Retrieving Folder Information 283
  Retrieving Additional Folder Information 285
  Retrieving Child Folder Information 286
  Creating Folders 287
  Modifying Folders 287
  Copying Folders 288
  Moving Folders 289
  Deleting Folders 289
  Emptying Folders 289
Conversation Operations 290
  Retrieving Conversations 290
  Performing Actions on Conversation 292
Availability Operations 293
  Find User Availability 294
  Get Room List 294
  Get Room 295
Improved Integration with Microsoft Office Documents 295

CHAPTER 18 Apache Solr 297
Say Goodbye to Verity 297
Updates to <cfindex> 297
  Boosting Fields and Documents 302
  Ordering Results 304
Data Import Handlers 304
  Setup 305
  Indexing Data 307

CHAPTER 19 Miscellaneous Enhancements 309
Each Release Brings Many Enhancements 309
Architectural Enhancements 310
  Verity-Solr Migration 310
  ColdFusion 9.0.2 Replaces 9.0 310
  Multiserver Enhancements 311
CFML Enhancements 312
Administrator Enhancements 314
  Some Modest Administrator Enhancements 314
  Only One Administrator Logon at a Time 315
  Admin API Enhancements 315
Security Enhancements 315
  Single-User CFLOGIN 315
  Sandbox Permissions for RDS 315
  Backward-Compatibility Issues 316
Logging Enhancements 317
  Log Locations 317
  New Metrics Logging 317
  New Request Logging (Access Logs) 318
  Solr Access Logs 318
Updated Embedded Libraries 319
  Web Services and Axis 319
  XML Support 319
  Underlying Java Architecture 319
  Other Libraries 320
Other Enhancements 320
  Query-Caching Enhancements 320
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual File System Enhancements</td>
<td>320</td>
</tr>
<tr>
<td>Solr Enhancements</td>
<td>321</td>
</tr>
<tr>
<td>Differences Between Enterprise and Standard Editions</td>
<td>321</td>
</tr>
<tr>
<td>Developer Edition Restrictions Eased</td>
<td>321</td>
</tr>
<tr>
<td>Licensing</td>
<td>322</td>
</tr>
<tr>
<td>Other Resources</td>
<td>322</td>
</tr>
<tr>
<td>Adobe ColdFusion 10 Documentation</td>
<td>322</td>
</tr>
<tr>
<td>Adobe ColdFusion Blog</td>
<td>322</td>
</tr>
<tr>
<td>Adobe ColdFusion Developer Center</td>
<td>323</td>
</tr>
<tr>
<td>“Adobe ColdFusion 10 Tutorials and Resources”</td>
<td>323</td>
</tr>
</tbody>
</table>

*Index* 325
INTRODUCTION

What Is This Book?
ColdFusion needs no introduction: It helped usher in the era of web-based applications over a decade and a half ago, and it remains an innovator in this space to this day. With each update, ColdFusion has further empowered us to build and create the ultimate online experiences, and ColdFusion 10 is no exception.

ColdFusion 10 is indeed a very important release: one that builds on the success of ColdFusion 9 by adding invaluable new features and functions. And that is key: ColdFusion 10 does not change much about the previous release; it adds features and functions. This means that ColdFusion 9 code and applications should run just as is in ColdFusion 10, and any books and tutorials on ColdFusion 9 apply to ColdFusion 10 as well.

And this presented the publishers and authors with a dilemma. ColdFusion Web Application Construction Kit (affectionately known as CFWACK) started off as a single volume, and then grew to two volumes in ColdFusion 4, and has been three volumes since ColdFusion 8. Recognizing that so much of the existing content for ColdFusion 9 applied as-is to ColdFusion 10, we could not in good conscience justify updating all the books and making readers buy them all over again. Plus, to make room to cover the new features in ColdFusion 10, we would have needed to remove chapters from the existing books, and as ColdFusion’s breadth and scope has increased, removing content has proven to be a difficult task.

After lengthy discussions with the publisher, the ColdFusion product team, and the authors, we opted not to update the three CFWACK volumes for ColdFusion 10. Instead, we decided to create a fourth volume to focus exclusively on what is new and improved in ColdFusion 10. And this is the book you are now holding in your hands.

This book, and indeed the entire ColdFusion Web Application Construction Kit series, is written for anyone who wants to create cutting-edge web-based applications.

If you are just starting creating your web presence, but know you want to serve more than just static information, this book, in conjunction with the ColdFusion 9 books, will help get you there. If you are a webmaster or web page designer and want to create dynamic, data-driven web pages, this book is for you as well. If you are an experienced database administrator who wants to take advantage of the web to publish or collect data, this book is for you, too. If you have used ColdFusion before and want to learn what’s new in ColdFusion 10, this book is also for you. Even if you are an experienced ColdFusion user, this book provides you with invaluable tips and tricks and serves as a definitive ColdFusion developer’s reference.

This book teaches you how to create real-world applications that solve real-world problems. Along the way, you acquire all the skills you need to design, implement, test, and roll out world-class applications.
How to Use This Book

As already noted, this book is designed to extend and enhance the three existing *ColdFusion 9 Web Application Construction Kit*. The four books are organized as follows:


- **Volume 2**—*Adobe ColdFusion 9 Web Application Construction Kit, Volume 2: Application Development* (ISBN 0-321-67919-9) contains Chapters 22–45 and covers the ColdFusion features and language elements that are used by most ColdFusion developers most of the time.

- **Volume 3**—*Adobe ColdFusion 9 Web Application Construction Kit, Volume 3: Advanced Application Development* (ISBN 0-321-67920-2) contains Chapters 46–71 and covers the more advanced ColdFusion functions, including extensibility features, as well as security and management features that will be of interest primarily to those responsible for larger and more critical applications.

- **Adobe ColdFusion 10 Web Application Construction Kit, ColdFusion 10 Enhancements and Improvements** (ISBN 0-321-89096-5) contains 19 new chapters focusing on what’s new in ColdFusion 10, and is intended to be used in conjunction with the three other titles listed here.

These books are designed to serve two different, but complementary, purposes.

First, as the books used by most ColdFusion developers, they are a complete tutorial covering everything you need to know to harness ColdFusion's power. As such, the books are divided into parts, or sections, and each section introduces new topics building on what has been discussed in prior sections. Ideally, you will work through these sections in order, starting with ColdFusion basics and then moving on to advanced topics. This is especially true for the first two books.

Second, the books are invaluable desktop references. The appendixes and accompanying website contain reference chapters that will be of use to you while developing ColdFusion applications. Those reference chapters are cross-referenced to the appropriate tutorial sections, so that step-by-step information is always readily available to you.

The following sections describe the contents of this new volume

**Part I: Web Technology Innovation**

ColdFusion is predominantly used to power web applications, and so Part I of this book focuses on ColdFusion web technology enhancements and innovations:

- Chapter 1, “ColdFusion 10 and HTML5,” introduces new HTML5 tags and features and explains how these are supported by ColdFusion.
Chapter 2, “Using WebSocket,” introduces an additional HTML5 enhancement, WebSocket, and explores the ways that it changes server-to-browser communication.

Charting and graphing has long been a ColdFusion staple, and Chapter 3, “Charting Revisited,” introduces new HTML5 charting options.

In Chapter 4, “Web Services,” you learn how to implement and use the latest generation of web services technologies.

The web services discussion continues in Chapter 5, “Using REST Web Services,” which introduces REST as an alternative to traditional web services.

Extensive video support is new to ColdFusion 10, and Chapter 6, “Embedding Video,” explains what you can do and provides useful tips and tricks.

**Part II: (Even More) Rapid Development**

ColdFusion has always been about productivity and rapid development, and ColdFusion 10 continues to innovate to help make developers even more productive:

- The CFML language is the magic that makes ColdFusion coding so productive and simple, but that simplicity need not be at the expense of power and flexibility. Chapter 7, “CFML Enhancements,” introduces closures and additional CFML enhancements.

- CFScript is the scripting alternative to CFML, and it too gains new features in ColdFusion 10, as discussed in Chapter 8, “CFScript Enhancements.”

- Chapter 9, “Object Relational Mapping Enhancements,” focuses on what’s new in ColdFusion’s ORM support, including new search options, HQL logging, and more.

- ColdFusion has been built on Java since ColdFusion MX (aka ColdFusion 6), and experienced ColdFusion developers quickly learn how to leverage the underlying Java engine to build more powerful applications. Chapter 10, “Enhanced Java Integration,” teaches you how to use custom class paths, dynamic proxies, and more.

- Chapter 11, “XML Enhancements,” explains the new and improved XPath support, which provides greater XML processing flexibility.

**Part III: Enterprise Ready**

Some of the most significant enhancements in ColdFusion 10 focus on administration and security:

- In Chapter 12, “ColdFusion in the Cloud,” you learn how to take advantage of innovations in cloud computing, and ColdFusion’s new support for simplified cloud deployment and hosting.

- Chapter 13, “Improved Administration,” teaches you all you need to know about the updated ColdFusion Administrator, including the critically important hotfix and update installer.
Chapter 14, “Scheduling,” introduces the new and improved scheduler and explains grouping, prioritization, event chaining, and more.

Chapter 15, “Security Enhancements,” introduces the latest security risks and concepts and explains which you should worry about and why.

Caching has always been an important part of performance optimization, and Chapter 16, “Improving Performance,” presents the latest and greatest caching techniques.

Microsoft Exchange and Microsoft Office are critical to most organizations, and Chapter 17, “Improved Integration,” focuses on integration with the latest versions of these applications.

Apache Solr was introduced in ColdFusion 9 as an alternative to the existing full-text search engine. In ColdFusion 10, Solr is the default engine, and Chapter 18, “Apache Solr,” explains how to use this new technology.

In addition to everything covered thus far, ColdFusion 10 offers a long list of smaller (but no less useful) enhancements, as enumerated in Chapter 19, “Miscellaneous Enhancements.”

The Website
This book’s accompanying website contains everything you need to start writing ColdFusion applications, including:

- Links to obtain ColdFusion 10
- Links to obtain Adobe ColdFusion Builder
- Source code and databases for all the examples in this book
- An errata sheet, should one be required
- An online discussion forum

The book web page is at http://www.forta.com/books/0321890965/.

And with that, turn the page and start reading. In no time, you’ll be creating powerful applications powered by ColdFusion 10.
Integration has always been a key strength and an important focus for ColdFusion. ColdFusion supports most of the messaging frameworks and protocols required to effectively communicate with different platforms, and web services are no exception.

On the basis of architectural style, two primary categories of web services are available: traditional Simple Object Access Protocol (SOAP)-based services and Representational State Transfer (REST)-compliant services. ColdFusion has long made it easy to create and consume SOAP-based web services. ColdFusion 10 builds on this foundation by upgrading the underlying engine for SOAP-based web services and supporting creation of REST-based web services. In this chapter we focus primarily on SOAP-based web services. Chapter 5 covers REST-based web services in detail.

**NOTE**

Web services support in ColdFusion 10 builds on that of prior versions. Basic web services support is covered extensively in Adobe ColdFusion 9 Web Application Construction Kit, Volume 3: Advanced Application Development, in Chapter 59, “Creating and Consuming Web Services.”

### What Are Web Services?

A web service is a web-based application or a network-accessible interface that can communicate and exchange data with other such applications over the Internet without regard for application, platform, syntax, or architecture.

At its core, it’s a messaging framework built on open standards through which different applications interact with each other over the Internet, abiding by a definite contract. As simple as it may sound, it truly enabled disparate applications to seamlessly interact and provide integrated solutions. It streamlined the integration of new applications among vendors, partners, and customers without the need for the centralized or proprietary software of enterprise application integration (EAI).
There are many reasons for the success of the web services framework, but these are the most important:

- **Standard communication**: It is based on widely adopted open standards such as Hypertext Transfer Protocol (HTTP) and Extensible Markup Language (XML).

- **Platform independence**: It provides an unambiguous way of defining, invoking, and consuming a service through Web Services Description Language (WSDL) and SOAP.

- **Loose coupling**: Web services components are loosely coupled. A web service requires almost no knowledge of the definitions of other separate components.

- **Effortless integration**: It's widely adopted and has great tooling support. Creating, publishing, and consuming a web service on any platform is basically like creating any other component, with no additional effort.

For now, don’t worry about all the acronyms referred to here; we will look at them closely in subsequent sections.

### A Typical Web Service Invocation

Let’s now look at the steps involved in a complete web service invocation and see how the process works (Figure 4.1).

![Figure 4.1](image)

**Typical web service invocation.**

1. (Optional) If a client has no knowledge of a web service it is going to invoke, it may ask a discovery service (usually on a separate server) to provide a service registered with it that meets the client’s requirements. However, in most cases, clients will know which service they want to invoke; hence, this step is optional.
2. (Optional) On receiving an inquiry, the discovery service will return the available service details registered with it.

3. The client needs to know how to invoke the service. It asks the web service to describe itself.

4. The web service replies with an XML document describing the methods, arguments, and types in WSDL.

5. Now that the client knows where the service is located and how to invoke it, using the WSDL provided the web service creates the artifacts necessary to invoke and consume the web service. It then sends a request over HTTP with all the details required in its body in SOAP format.

6. The web service returns the result of the operation in a response over HTTP, again in SOAP format.

**Web Services Architecture**

Now let’s look at the web services architecture and the key technologies that make web services possible. For completeness, Figure 4.2 shows the main components of the architecture.

A web service uses a transport medium such as HTTP to send messages in SOAP format, which is based on XML, abiding by the contract between a client and a server described using WSDL.

Several technologies are absolutely central to the distributed architecture of web services. Among these are HTTP, XML, and SOAP. Additionally, WSDL, a descendent technology, standardizes the syntax for describing a web service and its operations.

**HTTP**

HTTP is a communications protocol for exchanging information over the Internet. It is the common transport mechanism that allows web service providers and consumers to communicate.

**XML**

XML is similar to HTML in that it uses tags to describe and encode information for transmission over the Internet. HTML has preset tags that define the way that information is displayed. XML lets you create your own tags to represent not only data but also a multitude of data types, which helps ensure accurate data transmission among web service providers and consumers.
SOAP
SOAP is a lightweight protocol for the exchange of information in a distributed environment. SOAP can be used in messaging systems or for invoking remote procedure calls. It is based on XML and consists of three logical parts:

- A framework for describing what is in a message and how to process it
- A set of encoding rules for interpreting application-defined data types
- A convention for representing remote procedure calls and responses

SOAP handles the onerous job of translating data and converting data types between consumers and web service providers.

Currently, two versions of SOAP are available: Version 1.1, and Version 1.2, which was an upgrade of Version 1.1. Both are World Wide Web Consortium (W3C) standards, and web services can be deployed using either version. However, Version 1.2 offers some significant advantages over its predecessor, as you will see in the coming section.

**NOTE**
To learn more about the SOAP specification, see the W3C’s note about SOAP at http://www.w3.org/TR/soap.

WSDL
WSDL is an XML-based language specification that defines web services and describes how to access them.

WSDL is used to explain the details needed to invoke a web service over the Internet. WSDL defines XML syntax for describing services between a set of endpoints: usually a client and a server that exchange messages. This documentation can then act as a road map for automating the details of a web service. WSDL describes the service interaction rather than the formats or network protocols used to communicate. It simply defines the endpoints and their data, regardless of the implementation detail.

Thankfully, today’s ColdFusion developers do not need to concern themselves with such intricacies, or with the need to write documentation by hand, because ColdFusion generates WSDL automatically. To view the generated WSDL for a ColdFusion component deployed as a web service, append the string ?wsdl1 to the component’s URL. The WSDL document is then displayed in your web browser.

**NOTE**
To learn more about the WSDL, see the W3C’s WSDL specification at http://www.w3.org/TR/wsd1.

The W3C’s Web Services Description Working Group has published a new specification, WSDL Version 2.0, which is based on SOAP Version 1.2. It is significantly different and not compatible with WSDL’s earlier specification (Version 1.1); hence, it is called Version 2.0, not Version 1.2. We will look at the differences between the two versions and the advantages provided by WSDL 2.0 in the next section.
ColdFusion Web Service Engine

ColdFusion uses Apache Axis, a proven and reliable implementation of SOAP, as its underlying engine to implement SOAP-based web services. ColdFusion Version 9 uses Axis 1, which supports only WSDL 1.1 and SOAP 1.1. Therefore, ColdFusion 9 cannot create web services using these specifications and cannot consume external web services that used newer versions of WSDL and SOAP.

ColdFusion 10 has upgraded its web services support and includes implementation based on Axis 2 as well. ColdFusion now supports both versions of the WSDL and SOAP specifications simultaneously while maintaining backward compatibility.

We have already noted that support for newer specifications is important for ColdFusion to play well with external web services that use these specifications. This support also provides other benefits, discussed in upcoming sections.

In the subsequent sections, we will treat Apache Axis as a web service engine.

SOAP 1.2

SOAP 1.2 can do everything that SOAP 1.1 does and more. Among others the most notable differences between the two versions are the following:

- **SOAP 1.1 is based on XML 1.0, and SOAP 1.2 is based on XML Information Set (XML Infoset).** XML Infoset provides a way to describe the XML document with an XSD schema but is not bound to serialize using XML 1.0 serialization. Thus, SOAP 1.2 places no restriction on the way that the Infoset data is transported.

- **SOAP 1.2 provides a binding framework.** You can use SOAP 1.2’s specification of a binding to an underlying protocol to determine which XML serialization is used in the underlying protocol data units, thus making SOAP truly protocol independent.

- **SOAP 1.2 includes HTTP binding.** It provides support for both HTTP GET and POST operations and conforms better to web architectural principles. Thus, it uses established web technologies for improved performance.

- **SOAP 1.2 provides a clear processing model.** SOAP 1.2 is more robust and less ambiguous because it has resolved many of the interoperability issues that were concerns with SOAP 1.1.

To conclude, SOAP 1.2 is truly protocol agnostic, extensible, unambiguous, and more HTTP friendly than SOAP 1.1.

**TIP**

To visually differentiate between SOAP 1.2 and SOAP 1.1, look for xmlns:soapenv in the soap message. SOAP 1.2 will use http://www.w3.org/2003/05/soap-envelope, and SOAP 1.1 will use http://schemas.xmlsoap.org/soap/envelope/.
WSDL 2.0

WSDL 2.0 is the newer version of WSDL and a W3C standard. It is significantly different from WSDL 1.1 and provides additional benefits since it:

- **Uses SOAP 1.2:** WSDL 2.0 uses SOAP 1.2 to provide all the benefits of SOAP 1.2 such as better extensibility.

- **Supports interface inheritance:** As with Object-Oriented Programming (OOP), a WSDL document can inherit from another WSDL document, providing reusability and component-based architecture.

- **Supports additional schemas:** Along with the XML schema, WSDL 2.0 supports the use of RelaxNG and DTD as type definitions.

- **Supports additional message patterns:** WSDL 2.0 supports eight new patterns, including only single messages.

Although it is an enhanced version, WSDL 2.0 is mainly used to create SOAP-based REST-complaint web services. One reason for its lower rate of adoption is that WSDL 1.1 is sufficient for most common needs of most applications.

To view the generated WSDL 2.0 for a ColdFusion component deployed as a web service, append the string ?wsdl2 to the component’s URL. Note that WSDL 2.0 is available only with Web Service Engine Version 2.

Building Your First Web Service

Now it’s time to get started building a web service with ColdFusion.

To build a web service in ColdFusion, all we need to do is to write a ColdFusion component (CFC). CFCs take an object-like approach to the grouping of related functions and encapsulation of business logic. They also play a pivotal role in defining and accessing a web service.

We can create or reuse a prebuilt CFC with the operations we want to expose as functions and make the CFC accessible to remote calls by specifying `access="remote"`. The CFC location then becomes the endpoint for the web service, and its remote functions become operations that can be invoked on this web service.

**NOTE**

For more information about CFCs, read Chapter 24, “Creating Advanced ColdFusion Components,” in Adobe ColdFusion 9 Web Application Construction Kit, Volume 2: Application Development.

Now let’s create our first web service. We’ll start with the component in Listing 4.1.
Building Your First Web Service

Listing 4.1  /cfwack/4/hello.cfc

```cfc
<cfcomponent>
    <cffunction name="helloWorld" returnType="string" access="remote">
        <cfreturn "Hello World!">
    </cffunction>
</cfcomponent>
```

The `hello` CFC starts with a `<cfcomponent>` tag, which wraps the component’s content. Then the `<cffunction>` tag with a name and return type defines a single function, which simply returns a static string. The optional access attribute is set to `remote`, which exposes this CFC as a web service. And now we have a simple CFC-based web service, which we can publish and allow to be called by web service clients across the web.

To quickly verify that this web service works, open http://localhost:8500/cfwack/4/hello.cfc?wsdl in a browser. It should output the generated WSDL. Listing 4.2 shows part of the generated WSDL for this `hello` CFC.

Listing 4.2  Part of the WSDL Generated for hello.cfc

```xml
<!-- Some attributes have been omitted to keep focus only on elements relevant to our discussion later in this section-->
<wSDL:binding name="cfwack.4.hello.cfcSoap12Binding" ...>
    <soap12:binding ... style="document"/>
    <wSDL:operation name="helloWorld">
    <soap12:operation ... style="document"/>
    <wSDL:input>
    <soap12:body use="literal"/>
    </wSDL:input>
    <wSDL:output>
    <soap12:body use="literal"/>
    </wSDL:output>
    <wSDL:fault ...>
    <soap12:fault use="literal" .../>
    </wSDL:fault>
    </wSDL:operation>
</wSDL:binding>
<wSDL:service name="cfwack.4.hello.cfc">
    <wSDL:port name="cfwack.4.hello.cfcHttpSoap11Endpoint" ...>
    <wSDL:port name="cfwack.4.hello.cfcHttpSoap12Endpoint" ...>
</wSDL:service>
```

Here are a few details that you will observe here:

- An operation "helloWorld" is listed with the same name as that of the `<cffunction>` function.
- The WSDL has all the information such as types, endpoints, and message style required to execute this operation.
- Both SOAP 1.1 and SOAP 1.2 endpoints are supported through the same WSDL.
- The default WSDL style generated is Document Literal. You can change this style, as you will see later in this chapter.
ColdFusion will additionally log the web service engine used to deploy and access this web service on the console. This log can come in handy when you want to verify that the WSDL was refreshed and identify which web service engine was used for that particular operation.

**Consuming a Web Service**

Next we will see how to invoke this web service from ColdFusion. There are several ways to invoke any web service in ColdFusion, and we will look at them one by one.

Let’s start with the `<cfinvoke>` tag. This tag can be used to invoke a web service by specifying the `webservice` attribute, as shown in Listing 4.3.

**Listing 4.3** Calling a Web Service with `<cfinvoke>`

```cfml
<cfinvoke
    webservice = "#wsURL#"
    method = "helloWorld"
    returnVariable = "result">
<cfoutput><H1>#result#</H1></cfoutput>
```

Here we are trying to invoke the `hello` CFC that we wrote earlier, using the `<cfinvoke>` tag. We specified the CFC’s WSDL using the `webservice` attribute, the method to be invoked as “`helloWorld`” uses the same name as the function, and `returnVariable` is used to store the result of this operation. Then we simply output the result using `<cfoutput>` wrapped with HTML `<H1>` tags. If you run this code in a browser, you should get output similar to that in Figure 4.3.

**Figure 4.3**

Output generated from `hello` CFC.

Hello World!

Also, since web services are built on top of HTTP, you may need to provide other attributes such as proxy details if you are using an HTTP proxy server to connect to the web service provider and passwords similar to those for the `<cfhttp>` tag for calls using the `<cfinvoke>` tag.

Similarly, you can use the `<cfobject>` tag to create a web service proxy object. Then you can invoke methods on this object, which will be delegated as web service calls to actual endpoints. Listing 4.4 uses `<cfobject>` to call the same `hello` web service described earlier, by specifying the type as `webservice`.

**Listing 4.4** Calling a Web Service with `<cfobject>`

```cfml
<cfobject
    name = "ws"
    webservice = "#wsURL#"
    type = "webservice">
<cfset result = ws.helloWorld()>
<H1><cfoutput>#result#</cfoutput></H1>
```
When in script mode—that is, within a `<cfscript>` block or when writing CFCs in script syntax—you can use the `CreateObject()` method to invoke a web service. It is the script equivalent of the `<cfobject>` tag. Listing 4.5 shows how to use `CreateObject` to invoke a web service.

**Listing 4.5  Calling a Web Service with CreateObject**

```cfscript
<cfscript>
    ws = CreateObject("webservice", wsURL);
    result = ws.helloWorld();
    writeoutput(result);
</cfscript>
```

**Refreshing Stubs**

Recall the steps we have discussed for calling a web service. Creation of a web service proxy with `<cfobject>` or `CreateObject()` consists of steps 3 and 4 in Figure 4.1. Any call to a web service using this proxy will be steps 5 and 6. A web service call using `<cfinvoke>` involves steps 3 through 6.

However, steps 3 and 4 are computationally heavy operations and should be performed only once for a WSDL that is not changing. Hence, ColdFusion optimizes this operation significantly. ColdFusion makes a call to get the WSDL and generates the required stubs and artifacts only for the first call to the web service, using `<cfinvoke>` or `<cfobject>`. From the next call onward, it uses the generated stubs themselves, eliminating the need for steps 3 and 4.

This optimization creates a challenge in a development environment in which CFCs are being constantly modified. ColdFusion does not implicitly refresh the stubs for changed WSDL. The clients themselves have to refresh these stubs every time the WSDL changes. This operation can be accomplished by setting the `refreshWSDL` attribute as `true` with `<cfinvoke>` or `<cfobject>`. Although in a production environment `refreshWSDL` should be `false`, this is its default value, too.

**Using Complex Data Types**

In this section, you will see how to work with different ColdFusion data types. For that purpose, let’s look at another CFC, this one with several functions that take some complex data types, such as a struct or query, as arguments (Listing 4.6). Here we will focus on two processes: how to pass an argument to a web service call and how to work on the result.

**Listing 4.6  /cfwack/4/complex.cfc**

```cfc
<cfcomponent hint="echoes back the input specified">
    <!---
    The purpose of these functions merely is to demo accepting
    and returning a complex object
    -->

    <cffunction name="echoStruct" returntype="struct" access="remote">
        <cfargument type="struct" name="argStruct"/>
    </cffunction>
</cfcomponent>
```
Listing 4.6 (continued)

(outputs argument passed to this function to console
good for debugging while developing the service
...>
<cfdump var="#argStruct#" output="console">
<!---- typically your logic goes here ---->
<cfreturn argStruct>
</cffunction>

<cffunction name="echoQuery" returntype="query" access="remote">
<cfargument type="query" name="argQuery"/>

<cfreturn argQuery>
</cffunction>

<cffunction name="echoDocument" returntype="xml" access="remote">
<cfargument type="xml" name="argDocument"/>

<cfreturn argDocument>
</cffunction>

<cffunction name="echoAny" returntype="any" access="remote">
<cfargument type="any" name="argAny"/>

<cfreturn argAny>
</cffunction>

</cfcomponent>

Let’s analyze what’s going on here. We have created a complex CFC that isn’t actually very complicated. As you can see, it simply takes native ColdFusion data types as arguments and returns them back: the function echoStruct takes a struct as an argument, the function echoQuery takes a query as an argument, and the function echoAny can take any type as an argument.

Note that no special treatment is required to handle web service calls, and this CFC can work directly by creating its object and can also serve Ajax Remoting and Adobe Flash Remoting calls. You get the same data types to work with. It is ColdFusion’s responsibility to internally serialize the given data type to XML format before sending the web service call as a client, and to deserialize this XML back to the desired data type and pass it as an argument to the invoked function when it receives the web service to process it as a server.

Passing Arguments

Next, let’s see how to invoke this web service with ColdFusion. We have already talked about different ways to invoke web services, and for this example we will use `<cfinvoke>`. Let’s look at how to pass an argument and work with the returned object (Listing 4.7).
As shown here, there are three ways to pass arguments to web service calls. Let’s look at them one by one.

First we will see how to pass an argument using the `<cfinvokeparam>` tag. We begin by creating the WSDL URL to be passed with `<cfinvoke>`. We then create a struct with implicit notation. And in case the syntax confuses you, ColdFusion also supports JavaScript-style syntax for declaring the struct. Next we use `<cfinvoke>` to call the web service by using `<cfinvokeparam>` as its child tag and passing the arguments specified as a key-value pair. This key will be matched with the arguments declared in the function and will be populated likewise.

Alternatively, you can pass arguments as superfluous attributes in the `<cfinvoke>` tag itself, as shown in next call to `echoQuery`. Again, the attribute name is the argument name, and its value is the value that we want to pass to the call.
Finally, you can also use argumentcollection to pass a struct with the argument name as the key and the value to be passed as its corresponding value as shown for the call echoAny call.

Note that you cannot use positional arguments with <cfinvoke> because the key is a mandatory attribute in all three scenarios described here. To use positional arguments, you can use <cfobject> or CreateObject() to generate a web service proxy and call methods on it. This call will behave similarly to any other method invocation for components and supports positional arguments, key-value syntax, and argument collection.

Also note that we passed a simple string to echoAny. We did this because the type definition of “any” in generated WSDL supports only simple data types. If you passed any complex object instead, it would fail with the “unknown type cannot serialize” exception.

We have now seen various ways to pass complex objects as arguments and get back complex objects as the result of that particular web service call. What’s interesting is that there is almost no difference between calling a web service and calling a component locally. That is the beauty of ColdFusion: the capability to abstract the hard wiring required to perform a complex task such as a web service call and expose it as something simple that we already know such as calling a component. This ease of use lets us focus on the actual business logic and not to be bothered with the underlying technology or mundane boilerplate code.

**Working with Multiple Arguments**

So far in our examples, we have seen functions with only one argument. But your real-world functions more likely will have more than one argument. And though the mechanism to call these functions as web services remains the same, there are a few details that you need to take care of.

When your function has more than one argument, you may want to make a few arguments required and the rest optional. However, the <cffunction> attribute Required is ignored when a CFC is called as a web service; for a web service, all arguments are required. ColdFusion doesn’t support method overloading, so in in cases in which you want to pass only a few arguments, you need to use either of two approaches:

- You can make the function private and define different public methods for all parameter combinations. These methods will internally invoke this private function within the CFC, which performs the actual processing and also honors the defaults.

- The second possible solution is to use a special value for arguments that you don’t want to pass: for example, NULL. Then within the function body, you can check IsNull() and place default values instead. If you use <cfinvoke>, then you can set the <cfinvokeargument> tag’s attribute omit as true. If you are using a proxy object created with <cfobject> or the CreateObject() method, you can simply pass NULL.

**Tip**

To create NULL in ColdFusion, you can either use javaCast( "null", 0 ) or call a function that returns nothing: for example, function null(){}. 


Securing Your Web Service

Security is a very important aspect to consider when developing your services. As more and more business functions are exposed as web services, the boundary of interaction keeps expanding, and so does your responsibility to address all security requirements such as authentication, access control, data integrity, and privacy. In this section, we explore some ways to secure your web services.

To begin, you can always publish your web service over HTTPS. This approach will guarantee point-to-point security because SSL secures communications at the transport level. However, these scheme has limitations such as scalability issues, and you may not be able to use it.

You can also use your web server to control access to the directories containing your web services, or you can use ColdFusion security in the same way that you use it to control access to any ColdFusion page. The `<cfinvoke>` tag includes the `username` and `password` attributes that let you pass login information to a web server using HTTP basic authentication.

Using ColdFusion to Control Access

Let’s look at how to secure our web services from within ColdFusion. There are many possible ways to do so, and we will discuss just some of them here. You can pick the approach best suited to your particular needs.

One scheme that you can use uses `<cflogin>`. Rather than letting web servers handle authorization, you can implement authentication at the application level with Application.cfm (Listing 4.8).

```cfm
<cfapplication name="wack4_secure">
  <cflogin>
    <cfset authorized = false>
    <!--- verify username and password --->
    <cfif isDefined("cflogin")
      and cflogin.name eq "foo"
      and cflogin.password eq "bar">
      <cfset authorized = true>
    </cfif>
  </cflogin>
  <cfif not authorized>
    <cfsetting enablecfoutputonly="yes"
      showdebugoutput="no">
    <cfheader statuscode="401">
    <cfheader name="WWW-Authenticate" value="Basic realm="Web Services"">
    <cfabort>
  </cfif>
</cfapplication>
```

This Application.cfm example includes a `<cflogin>` tag. As you may know, the body of this tag runs only if there is no logged-in user. Therefore, the example includes some logic to validate the user in the body of the `<cflogin>` tag. In a real-world scenario, you would be validating users against a data source, LDAP, and so on. If the check here fails, the request is simply aborted,
with a few authentication headers set. The same logic can be placed in the Application.cfc file’s method onRequestStart. This method is executed for all types of requests and is invoked before the actual call to the web service method is made.

You can also use `<cfloginuser>` from within the `<cflogin>` tag to identify an authenticated user to ColdFusion and specify the user ID and roles. This approach lets you set allowed roles in `<cffunction>`, which can invoke that function.

Next, we explore how to invoke the same old hello web service, using the code snippet shown in Listing 4.9.

**Listing 4.9 /cfwack/4/secureclient/basic.cfm**

```coldfusion

<cfinvoke webservice="#wsURL#" method="helloWorld" returnvariable="result" username="foo" password="bar">

<h1> <cfoutput>#result#</cfoutput> </h1>

As you can see, we are using the same `<cfinvoke>` tag that we have been using to additionally pass username and password information. This information will be populated in the `cflogin` struct, accessible within the `<cflogin>` tag as the name and password that we will use to validate our user.

Another approach is to use Open Standard for Authorization (OAuth) authentication. This authentication protocol allows applications to access a user’s data in a secure way. Several good libraries for both publishing and consuming OAuth integrations are available in ColdFusion for you to investigate.

Finally, you can use SOAP headers for authorization purposes: for example, you can set Web Service Security (WSS) headers and validate them either at the application level or the component level, as described in the next section.

**Working with SOAP Requests**

ColdFusion offers a variety of ways to work with the SOAP requests and responses involved in web services. Let’s look at them closely with an example.

We have already talked about CFCs and how the same components can be used to serve different types of requests such as Adobe Flash Remoting and Ajax Remoting calls. When you want to handle SOAP requests differently and to know whether the call originated as a web service call, you can use the function `IsSOAPRequest()`. This function will return true if the CFC is being called as a web service.

Also we have talked about how SOAP requests for web services use HTTP as the transport medium. You also may know that HTTP uses headers to pass additional workable information about the request and its response. So depending on the use case, you may need to read SOAP
Working with SOAP Requests

request headers—for example, to get the username—or add headers to your SOAP responses—for example, an authorization header. ColdFusion provides functions that let you read and add headers to your SOAP request or SOAP response. Listing 4.10 provides an example.

Listing 4.10  /cfwack/4/soap.cfc
<cfcomponent hint="Test for SOAP headers">

<cffunction name="test" returntype="string" access="remote">
    <cfset isSOAP = isSOAPRequest()>
    <cfif isSOAP>
        <!--- Get the first header as a string. --->
        <cfset username = getSOAPRequestHeader("http://somenamespace/", "username")>
        <cfset result = "username: " & username>
        <!--- Get the second header as a string. --->
        <cfset password = getSOAPRequestHeader("http://somenamespace/", "password")>
        <cfset result = result & " and password: " & password>
        <!--- Add a header as a string. --->
        <cfset addSOAPResponseHeader("http://somenamespace/", "returnheader", "AUTHORIZED", false)>
    <cfelse>
        <cfset result = "Not invoked as a web service">
    </cfif>
</cffunction>
</cfcomponent>

As you can see, we have a simple soap CFC that has only one function test. Within this function, we check whether the call is a SOAP request with isSOAPRequest(). If the result is true, we get the headers from the request—that is, username and password—using getSOAPRequestHeader() and set a returnheader header in the response using addSOAPResponseHeader(). If the result is false, then we send back the result “Not invoked as a web service.” This example is very simple, but you can see how different logic can be applied to perform various operations such as actual authentication.

Now let’s see this CFC in action by invoking the CFC once as a web service and for a second time as a local method on a component (Listing 4.11).

Listing 4.11  /cfwack/4/soap.cfm
<cfscript>
    ws = CreateObject("webservice", wsURL);

    // Set the username and passwordheader as a string.
    addSOAPRequestHeader(ws, "http://somenamespace/", "username", "user");
    addSOAPRequestHeader(ws, "http://somenamespace/", "password", "pass");
</cfscript>
Listing 4.11 (continued)

```cftagscript
// Invoke the web service operation.
result = ws.test();

// Get the first header as an object (string) and as XML.
header = getSOAPResponseHeader(ws, "http://somenamespace/", "returnheader");
</cftagscript>

<cfoutput>
SOAP Return value: #result#<br>
SOAP Header value: #header#<br>
</cfoutput>

<cfinvoke component="soap" method="test" returnvariable="result">
</cfinvoke>
<cfoutput>The cfinvoke tag returned: #result#</cfoutput>

Here we created a proxy for the SOAP web service using `CreateObject()`. Then we added two heads-
ers, `username` and `password`, for this proxy. These headers will be added to the SOAP request when
the actual call is made, which is when the test method is called. We get back the result of this web
service call in `result`. Additionally, we extract the response header that was set in the CFC from
the proxy object. Next, we call a function on the SOAP CFC directly and output its results.

The output of this template when executed in a browser is obvious. For the SOAP request, the
output will return the username and password that were sent as headers, and it will also return
`returnheader` as "AUTHORIZED". For a local call, it would simply return "Not invoked as a web service."

**Application Settings**

Four application-level web service settings can be used to apply certain properties related to web ser-
vice across a given application. These settings are defined in Application.cfc in the `this.wssettings`
struct, shown in Listing 4.12, which we will discuss one by one. Note that these application settings
are introduced in ColdFusion 10 and are not available in previous ColdFusion versions.

Listing 4.12  /cfwack/4/Application.cfc

```cfc
cfcomponent>

<cfset this.name="cfwack_4">

<cfset this.wssettings.version.publish = 2>
<cfset this.wssettings.version.consume = 2>
<cfset this.wssettings.style = "wrapped">
<cfset this.wssettings.includeCFTypesInWSDL = false>

</cfcomponent>
```

Application Settings

Four application-level web service settings can be used to apply certain properties related to web ser-
vice across a given application. These settings are defined in Application.cfc in the `this.wssettings`
struct, shown in Listing 4.12, which we will discuss one by one. Note that these application settings
are introduced in ColdFusion 10 and are not available in previous ColdFusion versions.
Including ColdFusion Types in WSDL

In our discussion of complex data types, we looked at the use of ColdFusion native data types with web services. These data types are similar to a number of data types in C++ and Java, but they do not exactly match any of the data types defined in the XML schema used by WSDL and SOAP for data-type representation and conversion.

This lack of a match is fine if a web service is published and consumed within ColdFusion because ColdFusion expects and understands its own data types and can serialize or deserialize them. However, when clients other than ColdFusion call this web service, they will need additional information to convert arguments to be sent with the web service call to data types that ColdFusion expects and understands.

Here is where `this.wssettings.includeCFTypesInWSDL` comes to our rescue. It tells ColdFusion whether to include ColdFusion’s native type information as an XML schema defined in the WSDL itself. Using this schema, other platforms can understand the arguments and result types.

If your web services will be used only with ColdFusion clients, there is no need to include this type information as it will increase the WSDL size. By default, it is set to `false`.

Deciding Which Web Service Engine to Use

Let’s step back a bit from our original topic of application settings. We said earlier that there are two web service engines available to us with ColdFusion 10. You can choose the web service engine to use according to your requirements:

- Web Service Engine Version 1: Use this version if you want to publish WSDL in RPC style, or if you do not want all your existing web service clients to refresh their generated stubs. You should also use this version when you have web service clients on ColdFusion 9 and you use complex data types.

- Web Service Engine Version 2: Use this version if you want to consume any web service that is based on WSDL 2.0, or if you want to publish WSDL in wrapped style. With ColdFusion 10, this is the default engine for publishing a web service.

- Either engine: For all other scenarios, you can use either of the web service engines.

Specifying the Web Service Engine

ColdFusion uses Web Service Engine Version 2 by default to publish any component as a web service. However, you can override this behavior and tell ColdFusion which engine to use. You can specify the web service engine used to publish your ColdFusion components in any of three places:

- Component level: You can specify `wsversion` with `<cfcomponent>` to declare the web service engine to use to publish that particular component. The possible values are 1 and 2. This setting takes the precedence over application- and server-level version declarations.
- **Application level**: You can specify `this.wssettings.version.publish` in your Application.cfm file to declare the web service engine at the application level. All the components in the application will then be published using this setting. This setting takes precedence over the server-level setting.

- **Server level**: You can specify the web service engine to be used across the server. Select the version on the Web Services page in the Data and Services section of ColdFusion Administrator, shown in Figure 4.4.

While consuming a web service, ColdFusion will try to understand the WSDL style. If the style is Document Literal or Document Literal Wrapped, ColdFusion automatically uses Web Service Engine Version 2, and if the style is RPC Literal, ColdFusion automatically uses Web Service Engine Version 1. However, the caller can override this behavior by specifying the web service engine to be used while consuming web services. There are two places to provide this option:

- **While consuming a service**: You can provide `wsversion` with `<cfinvoke>` to tell ColdFusion which web service engine to use to consume the web service. The possible values are 1 and 2, and this setting takes precedence over the application-level setting.

- **Application level**: You can specify `this.wssettings.version.consume` in your Application.cfm file. Any call to consume web services will now use the specified version of the web service engine.

### Choosing the WSDL Style

ColdFusion can publish WSDL and consume web services that publish WSDL in the following styles:

- **RPC Encoded**: Specified with the `<cfcomponent>` attribute `style="rpc"`, this style considers web services as XML-based forms of Remote Procedure Calls (RPCs). Here the SOAP message body contains only one element, which is named after the operation, and all parameters must be represented as subelements of this wrapper element. This style is available only with Web Service Engine Version 1.

- **Document Literal**: Specified with the `<cfcomponent>` attribute `style="document"`, this style considers web services as a means of moving XML information from one place to another. Here, the SOAP message body must follow the XML schema defined in WSDL as types. This style is available with both web service engine versions.

- **Document Literal Wrapped**: Specified with the `<cfcomponent>` attribute `style="wrapped"`, this style is similar to the Document Literal style, except that the SOAP message body is wrapped within a root element. This style is available only with Web Service Engine Version 2.
Alternatively, you can specify the WSDL style to use at the application level as `this.wssettings.style`. ColdFusion will use this information to generate WSDL in the specified style for all the CFCs in this application. You can individually override this setting with the `<cfcomponent>` attribute `style`, which takes precedence over application-level settings.

### Configuring Web Services in ColdFusion Administrator

The ColdFusion Administrator lets you register a web service with a name. You can do so by adding a web service on the Web Services page in the ColdFusion Administrator in the Data and Services section. When you reference that web service in your code with this name, you won’t have to specify the URL or any other details for the web service call. For example, any time you invoke a web service registered as `ZipCodeWS` on a particular server, you can refer to it as `WebService="ZipCodeWS"`. The URL can then be changed to point to another URL without the need to modify the invocation code throughout the application. This approach represents a type of code encapsulation, which you could also implement using application or request scope variables.

With ColdFusion 10, you can also specify proxy settings such as the proxy server, proxy port, proxy username, and proxy password, and a server-level setting to cause any web service request to time out at a particular time. When you call this web service by its name at the time of registration, you need not specify these settings again. However, settings provided at the time of the actual web service call, such as a `<cfinvoke>` call, will override server-level settings.

Note that just accessing any web service from user code will not autoregister or change that web service in ColdFusion Administrator, which used to happen until ColdFusion 9. With ColdFusion 10, to register a web service in ColdFusion Administrator you need to add or modify it from the Administrator only.

**NOTE**

To learn more about ColdFusion Administrator changes, visit http://www.adobe.com/devnet/coldfusion/articles/axis2-web-services.html.

### Best Practices

Web services have been around for a while and have generated significant hype. Along with the advantages of cross-platform compatibility are some drawbacks. Although the distributed computing environment of web services is widely recognized as the way of the future, it carries the baggage of network latency and additional translation time. The actual overhead of running a web service is not as bad as perceived, but it is a factor to consider when selecting parts of systems to expose to the world. Careful testing and optimization can reduce this potential problem significantly. Here are several general principles to consider when programming and designing web services:

- Use coarse-grained web services. Network latency can be the biggest performance bottleneck. Try to reduce calls to the server. Call a web service once and use a query of queries to return the detailed information for display.
- Secure your services. Never publish your web service without proper security in place. See the discussion about securing your web services earlier in this chapter for details.

- Use a server-level timeout. Aim for a timeout value of 1 to 3 seconds; waiting for a web service from a busy server to return can eat up all threads on your server and potentially can bring it down.

- Preferably, call long running web services from a scheduler or `<cfthread>` and look for caching possibilities. See Chapter 14 for information about schedulers and Chapter 16 for information about caching.

- Use stateless web services whenever possible.

- Include ColdFusion types in WSDL and limit the use of complex data types in web services that interact with other platforms. Other platforms may not be able to understand deeply nested types.

- Monitor your web service calls to understand their use. You can use ColdFusion server monitoring, which monitors web service calls separately from other types of request. With just the basic monitoring enabled, you can get valuable information about web service requests, running or queued. And on the basis of this information, you can tweak your server settings and also change the application code.

**Troubleshooting**

No matter how carefully you code, you can always end up with unexpected results. In such cases, you will want to see what is happening and to pinpoint the code that is causing the problem. Debugging can be difficult when there is client-server communication as in the case of web services. Here are a few tips to help you identify and fix common problems that you may face:

- Check the WSDL. As explained earlier, append `?wsdl` to the CFC URL and run the URL in a browser to see whether the CFC has any compilation problems and whether the generated WSDL is correct and accessible.

- Use `<cfdump>` to output to the console to check whether the call is coming to your application and see what arguments are being passed. Remember to remove this function when implementing your application for production.

- Use `refreshWSDL`. It is possible that the CFC you are accessing through the web service has changed. Use this attribute with `<cfinvoke>` to regenerate the stubs. Remember to remove it when implementing your application for production.

- If `wsversion` is not defined while consuming a web service, ColdFusion checks the WSDL to determine which web service engine it should use to consume that particular web service. You can force ColdFusion to use a specific web service engine by specifying `wsversion` with `<cfinvoke>`.
You can use the `GetSOAPRequest` function to get the actual SOAP request sent and the `GetSOAPResponse` function to get the actual response received. This information can help you determine whether correct information is being sent and whether you are receiving the correct response.

You can use a TCP monitor such as TCPMon to track exactly what is being sent and received over the wire. The monitor acts like a proxy between the client and the server and shows you the communication that occurred in between them.

**SOAP or REST?**

If you have ever wondered which form of web services you should use, you are not alone. However, there is no easy answer. In this section, we list the key differences between SOAP and REST services to help you decide which style to choose for your particular case.

- SOAP-based web services are object oriented, and REST-based web services are representation oriented. Without going into detail, this distinction means that you can get started easily with a SOAP-based solution, but a REST-based solution will need additional planning to create a logical hierarchy.

- SOAP-based web services are declarative, use the standard WSDL format to describe them, and have great tooling support. REST-based web services do not yet have a standard for describing services.

- SOAP-based web services support only XML, and REST-based web services can support numerous content types, including JavaScript Object Notation (JSON), and can be accessed directly from JavaScript.

- SOAP 1.1–based solutions do not conform to the HTTP model and hence cannot take advantage of HTTP caching, security, and so on. REST is fully HTTP complaint.

- XML use makes the SOAP format verbose and its performance slower than REST using JSON. However, REST clients may take a longer time when using XML.

- REST generates search-engine optimization (SEO)–friendly endpoints.

As a general rule, REST benefits web services directly accessed from web pages as in the case of `XMLHttpRequest` (XHR), and SOAP benefits web services accessed by an intermediate server or middleware.
This page intentionally left blank
This page intentionally left blank
INDEX

SYMBOLS AND NUMBERS
: (colon separator), in structures, 128–129
= (equals), for assignment, 129
[ ] (square braces), for use with arrays, 129
{} (curly braces), use around structure keys, 129
2D graphics, what's new in HTML5, 3

A
Access control
resource and sandbox security, 251
restricting by IP address, 221
web services, 67–68
Action attribute, <cfschedule>
with “List” value for listing tasks, 243–245
with “pause” or “resume” values, 246
Add/Edit scheduled tasks, 247
Admin API enhancements, 315
Administration enhancements. see also ColdFusion
Administrator
caching updates, 219
logging changes, 219–220
new theme, 217–218
overview of, 217
password reset script, 220
RDS configuration, 220
restricting access by IP address, 221
security updates, 219
server updates, 221–224
Adobe ColdFusion Developer Center, 323
Adobe Flash
automatic fallback for browsers that do not support
WebSocket, 10, 96
error handling and unsupported browsers and, 34
formats supported by Flash Player, 93
playing video with, 93
playlist support, 100
remoting, 64
rendering charts and, 36, 45
secure profiles, 252
video support in ColdFusion, 94–95
afterUnsubscribe method, in news handler
example, 18
Ajax
building detail page for chart, 46–48
new file and directory control, 218
remoting, 64
allowSubscribe method, in news handler example, 18
Alman, Ben, 107
Amazon CloudFront, 210
Amazon Elastic Beanstalk, 210
Amazon Machine Images. see AMIs (Amazon Machine Images)
Amazon Web Services. see AWS (Amazon Web Services)
AMIs (Amazon Machine Images)
publishing a public, 215–216
safely publishing, 215
selecting predefined, 205–206
Anklam, Ryan, 107
Apache Axis
SOAP implementation used by ColdFusion, 59
updating embedded libraries, 319
Apache Solr
access logs, 318
adding/maintaining collection data, 141
boosting fields and documents, 302–303
collection use in ColdFusion, 145
DIH (DataImportHandler), 304–308
enhancements, 321
filtering custom data, 300–302
managing collections, 137–140
ordering index results, 304
processing text indexes, 137
specifying custom fields, 297–300
updates to ColdFusion support, 295–296
updates to embedded libraries, 320
Verity-Solr migration, 310
Apple iTunes, 100
Apple Safari. see Safari
Application state, managing in cloud computing, 212
Application.cfc file
application-specific caching, 266–267
handling form fields with same name, 120–121
implicit notation and, 123
javaSettings property, 169
onAbort event added to, 119–120
ORM searches at application level, 159
Applications
advantages of cloud, 201
caching, 266–267
getApplicationMetaData() function, 313
ORM searches at application level, 159
scheduling tasks at application level, 226–228
securing, 252
web services at application level, 70
Architecture
- enhancements in ColdFusion release 9.0.2, 310–311
- multiserver deployments, 311–312
- Verity-Solr migration, 310
  of web services, 57–58
Arguments
- accepting Java arguments, 180
- creating functions as, 110–111
- event handling, 241
- passing, 134
- passing between web services, 64–66
- passing complex, 134–135
- working with multiple, 66
Arrays
- arrayEach example, 112
- arrayFilter example, 113–114
- arrayFind example, 112–113
- arraySort example, 114–115
- looping over Java arrays, 185–186
  for-in loops applied to Java arrays, 130
- serialization, 84
- square braces in syntax of, 129
Audio embedding, in HTML5, 3, 5
Audit logs, 219–220
Authenticate, JavaScript API function, 14
Authentication
- onWSAuthenticate method, 27–29
- securing web services, 67–68
  single sign-on, 30–31
Availability
- advantages of cloud, 201
  Microsoft Exchange Server, 293–295
avg function, XPath example, 189–190
AWS (Amazon Web Services)
- components of, 210
  EC2 (Elastic Cloud Compute), 204–207
  IaaS service model for, 202
- overview of, 203–204
  public cloud deployment, 203
  registering with, 204
  S3 (Simple Storage Service), 207–209
  service levels, 211–212
Axis. see Apache Axis

B
Backups, cloud computing and, 214
Backward-compatibility issues, 316–317
Bar charts
- example, 36
  types of, 41
beforeSendMessage method, in news handler example, 18
beforePublish method, in news handler example, 18
Bitmaps, in HTML5, 3
Blogs, ColdFusion, 322–323
Builder, ColdFusion, 220
Built-in functions, cache regions and, 269–271

C
Cache regions
- built-in functions and, 269–271
  overview of, 267–268
Caching
- application-specific, 266–267
- built-in cache functions, 269–271
- cache regions, 267–268
- EhCache version update, 275
- new cache functions, 271–272
- query caching, 273–275, 320
- Server Monitor support for, 275–277
- updates, 219
- what it is, 265–266
Calendars, Microsoft Exchange Server, 293–295, 313
Call stacks, debugging and, 313
Callback events, video embedding and, 97–98
canSendMessage method, in news handler example, 18
Canvas elements
- chart support and, 35
- rendering charts and, 45
- what’s new in HTML5, 4–5
Capability elasticity, features of cloud computing, 200
<cfabort>, onAbort events, 119–120, 313
<cfcache>. see also Caching
  caching functions, 270–271
  EhCache and, 265
<cfchart>. see also Charts/charting
  chart types, 40–41
  getting started with, 35–37
  overview of, 35
  plot attribute, 49–50
  previewing and zooming charts, 38–39
  renderer attribute, 45
  tags required for creating charts, 36
  updating, 37
<cfchartdata>, 36
<cfchartseries>
  plotting multiple series in charts, 45
  querying data in charts, 38
  tags required for creating charts, 36
<cfcollection>. see also Collections
  collection function in place of, 137–141
  creating collections, 305
  listing current collections, 298
<cfcomponent>, see also CFCs (ColdFusion components)
application-level settings for web services, 70
building web services, 61
event handling, 240
ORM searches at component level, 159
<cfcookie>, 132–133. see also Cookies
CFCProxy class
calling CFCs from Java, 175–176
constructors, 176–177
directInvoke argument of, 181
importing, 177–178
invoke method, 177
overview of, 176
reasons for using proxy features, 185
CFCs (ColdFusion components)
building REST CFCs, 79–80
building web services, 60–62
calling from Java, 175–176
collection component in place of CFCOLLECTION tag, 137–140
community-contributed tags, 146–147
CreateDynamicProxy function, 182–185
creating functions for, 111
documentation of, 145
errors returned from script-based tag-equivalents, 140–141
implicit constructors, 121–122
implicit notation, 123–124
index component in place of CFINDEX tag, 141–143
Invoking CFC methods from Java, 179
metadata, 146
method chaining, 122–123
newsHandler example, 17–19
ORM searches, 159
proxying as Java objects, 182–185
reasons for using REST services and, 77
reusing CFC instances, 135–136
search component in place of CFSEARCH tag, 143–145
serialization, 84
source code for, 146
tag equivalents, 147
testing REST CFCs, 80–82
updates to, 121
what you can do with CFC handlers, 16
<cfdump>
show attribute, 144
troubleshooting web services, 74
<cfexchangecalendar>, 294–295, 313. see also
Calendars, Microsoft Exchange Server
<cfexchangeconversation>, 290–293, 313. see also
Conversations, Microsoft Outlook
<cfexchangefolder>. see also Folders, Microsoft
Exchange Server
copy action, 288
create action for creating folders, 287
enhancements, 313
findSubFolders action for retrieving child folder information, 286
getInfo action for retrieving folder information, 283–285
modify action, 287
move, delete, and empty actions, 289
<cffile>. see also files
strict attribute for file upload protection, 260
support for output content within tag body, 313
<cffunction>, specifying content types in REST, 82.
see also functions
<cfimage>, resize option, 312
<cfinclude>, runOnce attribute, 118–119, 312
<cfindex>. see also Indexes
DIH (DataImportHandler) features, 307–308
docboost and fieldboost attributes, 302–303
index component in place of, 141–143
orderby attribute, 304
specifying custom fields, 297–298
<cfinvoke>
authenticating web services and, 67–68
consuming web services, 62–63
passing arguments to web services, 64–66
<cflogin>
login security, 30–31
securing web services, 67–68
single user restriction in, 315
<cfloop>, looping over queries, 314
<cfmap>
attributes, 6
Cold Fusion 10 support for geolocation features of HTML5, 5–7
ShowUser attribute, 312
<cfmediaplayer>
callback functions, 97–98
DRM (digital rights management), 103
playing YouTube video, 101–102
repeat attribute, 98–99
style attribute, 97
title attribute, 99–100
type attribute for video support, 94–95
CFML (ColdFusion Markup Language)
arrays and inline functions, 112–115
calling Java program with CFML page, 180–181
closures, 107–112
component updates, 121
enhancements, 312–314
CFML (continued)

handling form fields with same name, 120–121
implicit constructors, 121–122
implicit notation, 123–124
Java integration, 165–168
lists and inline functions, 116
method chaining, 122–123
onAbort event added to Application.cfc file, 119–120
overview of, 107
passing CFCs to Java class, 184–185
queryNew and queryAddRow improvements, 124–125
runOnce attribute of <cfinclude>, 118–119
script-based development in, 128
source code for CFCs, 146
structs and inline functions, 116–118
syntax and behavior improvements, 118
WebSocket message integration, 16

<cfobject>
consuming web services, 62–63
loading Java classes, 167–168
<cfparam>, MAXLENGTH validation attribute, 313
<cfpop>, SECURE attribute for SSL processing, 313
<cfquery>, 273. see also Queries
<cfquery>, see also Scheduling
Action attribute with “List” value for listing tasks, 243–245
application-level tasks, 227
chaining tasks, 231–233
changes to scheduling, 226
clustering scheduled tasks, 249–250
cronTime attribute, 236–237
deleting tasks, 230
event handling, 239–243
Exclude attribute, 233–234
FIRE_NOW option for handling task misfires, 238–239
Group attribute, 228–229
onException attribute, 237–238
pausing/resuming tasks, 246
prioritizing tasks, 230–231
Repeat attribute, 234–235
running tasks, 230
<cfscript>
approaches to CFC tag equivalents, 147
calling Java program with CFML page, 180–181
collection component in place of CFCOLLECTION tag, 137–140
colon separator in structures, 128–129
community-contributed CFC-based tags, 146–147
consuming web services, 63
cookie statement in place of CFCOOKIE tag, 132–133
defining CFCs with, 179
documentation and source code of CFCs, 146
errors returned from script-based tag-equivalent CFCs, 140–141
evolution of script-based development in CFML, 128
index component in place of CFINDEX tag, 141–143
invoke function in place of CFINVOKE tag, 133–136
looping, 129–130
looping over Java arrays, 185
overview of, 127
resources for, 147–148
search component in place of CFSEARCH tag, 143–145
setting statement in place of CFSETTING tag, 131–132
<cfsearch>
full-text searches, 150
performing basic search, 143–144
search component in place of, 143
search enhancements, 300–302
<cfsetting>, 131–132
Cfstat utility, secure profiles, 252
<cfthrow>, customizing responses in REST, 92
<cfwebsocket>. see WebSocket
Chaining scheduled tasks, 231–233
Channels, CFC handlers and, 17
Channels, WebSocket defining, 11
subchannels for message filtering, 23–25
Chart object, ZingChart, 45
Charts/charting
clickable URLs in, 45–48
getting started, 35–37
highlighting important data in, 49–50
overview of, 35
plotting multiple series in, 44
previewing and zooming, 38–39
querying data in, 38
refreshing, 51–53
rendering, 44
styles, 42–44
types, 40–41
updating, 37, 50–51
Checkboxes, handling form fields with same name, 120–121
Chrome
HTML5 support, 4
video support, 94
Classes, Java
benefits of dynamic loading, 172
challenges of adding .class files to loadPaths, 174–175
challenges of dynamic loading, 173–175
creating class for CFCs, 183–184
defining application-specific class libraries, 169
example of dynamic loading of Java classes, 171–172
loading, 168–169
naming load paths, 169–170
passing CFCs to, 184–185
reloading changed classes, 169, 172–173
unexpected class or version loaded, 175
using, 167
Clickjacking, 263
closeConnection() function, JavaScript API, 14
Closures
arrays and, 112–115
benefits of, 111–112
creating functions as arguments, 110–111
defined, 107
defining functions within existing functions, 108–109
lists and, 116
structs and, 116–118
UDF (user-defined function) example, 108
Cloud
advantages of, 201
Amazon Web Services. see AWS (Amazon Web Services)
cost model for, 211
deployment models, 203
design considerations, 212
designing for, 210
licensing and, 210–211
managing application state and transactions, 212
managing databases, 213–214
monitoring, 215
overview of, 199
publishing a public AMI, 215–216
security of, 214
service levels offered, 211–212
service models, 201–202
technologies of, 200–201
what it is, 199–200
CloudFront, Amazon, 210
Clustering scheduled tasks, 248–250
Code-level security, 251
ColdFusion Administrator
configuring web services, 73
DH (DataImportHandler) configuration, 305
enhancements. see Administration enhancements
refreshing REST services, 92
registering REST services, 78, 92
scheduling in, 247
secure profiles, 252
securing, 251
Server Monitor option, 275
WebSocket options, 34–35
ColdFusion Builder, 220
collection component
approaches to CFC tag equivalents, 147
creating/creating CFC tag equivalents, 147
creating CFC tag equivalents, 147
deleting collections, 140
listing collections with, 138
methods, 140
in place of CFCOLLECTION tag, 137
collections, Solr
adding/maintaining collection data, 141
collection use in ColdFusion, 145
managing collections, 137–140
processing text indexes, 137
Colon separator (:), in structures, 128–129
Communication standards, 56
Community cloud, 203
Compiling Java programs, 178–179
Complex data types, web services, 63–64
Components. see CFCs (ColdFusion components)
 Constructors
 CFCProxy class, 176–177
 implicit, 121–122
Content types, REST services, 82–83
Conversations, Microsoft Outlook
enhancements, 313
overview of, 290
performing actions, 292–293
retrieving, 290–291
cookie statement
approaches to CFC tag equivalents, 147
in place of cfccookie, 132–133
Cookies
secure profiles, 252
session management and, 261
Cost model, cloud computing, 211
CPUs
Amazon instance types and, 206
licensing and, 211
createDynamicProxy CFC function, 182–185
createObject() function, loading Java classes, 167–168
CreateObject() method
consuming web services, 62–63
creating proxy for SOAP web service, 70
Cron expressions, for Quartz scheduler, 235–237
cronTime attribute, <cfschedule>, 236–237
Cross-site scripting (XSS) protection, 252, 253–257
CSRF (cross-site request forgery) protection, 257–259
CSS3, what’s new in HTML5, 4
Curly braces ({}), used around structure keys, 129
Custom fields
boosting, 302–303
in indexes, 297–300
Customizing responses, with REST, 92
Data types
- complex in web services, 63–64
- including ColdFusion types in WSDL, 71
- specifying content types, 82–83
Databases
- creating shared tables when clustering scheduled tasks, 248–249
- managing in cloud computing, 213
DataImportHandler. see DIH (DataImportHandler)
Dates
- `dateTimeFormat()` function, 313
- in task scheduling, 233–234
Debugging, call stacks and, 313
Deployment models, cloud computing, 203
Derby, updates to embedded libraries, 320
Design considerations, cloud computing, 212
detail method, in REST application, 89
Developer edition, 321
Digital rights management (DRM), 103
DIH (DataImportHandler)
- indexing data, 307–308
- overview of, 304–305
- setting up, 305–306
directInvoke argument, of CFCProxy class, 181
Directories
- controlling access to, 251
DirectoryCopy( ) function, 313
DNS lookup, example of CFML/Java integration, 166–167
docboost attribute, `<cfindex>`, 302
Document Literal Wrapped, WSDL styles, 72
Document Literal, WSDL styles, 72
Documentation
- of CFCs, 146
- ColdFusion 10, 322
Documents
- docboost attribute, 302
- Microsoft Office. see Microsoft Office documents
DRM (digital rights management), 103
DTD, WSDL support for type definitions, 60

E
EAI (enterprise application integration), 55
EAR files, 312
EBS (elastic block storage), 206
EC2 (Elastic Cloud Compute)
- AMIs (Amazon Machine Images), 205–206
- instance types, 206
- overview of, 204
- public cloud deployment, 203
- purchasing model, 205
- region and availability zone, 205
- security types, 207
- service levels, 211–212
EhCache
- ColdFusion caching engine based on, 265
- `ehcache.xml` file, 266–267
- enhanced query caching and, 273–275
- updates to embedded libraries, 320
- version update, 275
Elastic Beanstalk, Amazon, 210
Elastic block storage (EBS), 206
Elastic Cloud Compute. see EC2 (Elastic Cloud Compute)
ELB (Elastic Load Balancer), from Amazon, 213
Email, notification of server updates, 224
Embedded libraries, updating
- Apache Solr, 320
- Derby, 320
- EhCache, 320
- Java, 319
- Lucene, 320
- overview of, 319
- Web services, 319
XML (Extensible Markup Language), 319
Embedding video. see Video embedding
Encoding functions, for XSS protection, 254–257
End-user license agreement (EULA),
- cloud computing and, 211
Enterprise edition
- application-level tasks, 226–227
- compared with Standard edition, 321
- scheduling in, 225
Ephemeral storage, EC2 storage options, 206
Equals (=), for assignment, 129
Error handling
- video embedding and, 97–98
- WebSocket, 34
EULA (end-user license agreement), cloud computing and, 211
Event handling
- arguments, 241
- `<cfcomponent>`, 240
- `<cfschedule>`, 239–240
- stubs, 240
- using, 241–243
EWS (Exchange Web Services), 281–282
Excel, Microsoft, 295–296
Exception handling, scheduled tasks and, 237–238
Exchange Server. see Microsoft Exchange Server
Exchange Web Services (EWS), 281–282
Exclude attribute, `<cfschedule>`, 233–234
Excluding dates, in task scheduling, 233–234
Explicit indexing, in ORM, 157–158
Extensible Markup Language. see XML (Extensible Markup Language)
Extensible Stylesheet Language Transformations. see XSLT (Extensible Stylesheet Language Transformations)

F
fieldboost attribute, <cfindex>, 302
Fielding, Roy, 77
Files
Expandpath() function, 313
location and arrangement in multiserver deployments, 311
upload protection, 260–261
FileZilla, 207
Filtering custom data, in indexes, 300–302
Filtering WebSocket messages overview of, 22
selectors, 25–27
subchannels, 23–25
FIRE_NOW option, <cfschedule>, 238–239
Firefox
HTML5 support, 4
video types supported, 94
Flash. see Adobe Flash
Flexibility, advantages of cloud, 201
folderPath attribute, <cfexchangefolder>, 285
Folders, Microsoft Exchange Server
creating, 287
enhancements, 313
modifying, 287–288
retrieving folder information, 283–286
for..in loops
looping over Java arrays, 130
looping over queries, 129–130
Forms, handling form fields with same name, 120–121
Full-text searches
with Apache Solr. see Apache Solr
of ORM entities. see ORM (object relational mapping)
Functions. see also by individual type
built-in, 269–271
caching functions, 270–272
creating as arguments, 110–111
for CSRF protection, 257–259
defining functions within existing functions, 108–109
encoding functions for XSS protection, 254–257
for extending SQL queries to ORM, 150
implementing tags as, 133
returning system-level information, 313
secure profiles, 252
uses of, 111

G
Garbage collection (GC), JVM tuning and, 277–279
GC (garbage collection), JVM tuning and, 277–279
Geolocation API, HTML5, 4–7
GetSOAPRequest function, troubleshooting web services, 75
getSubscriberCount, JavaScript API function, 14
getSubscriptions, JavaScript API function, 14
Google API key, 7–8
Google Chrome
HTML5 support, 4
video support, 94
Google maps
accessing, 7–8
Geolocation API and <cfmap> tag and, 5–7
Group attribute, <cfschedule>, 228–229
Grouping scheduled tasks, 228–229

H
Hash() function, for password security, 262
Heap, JVM tuning and, 278
Hibernate, ORM and, 149
Hibernate Query Language (HQL)
extending SQL queries to ORM, 150
logging, 162
Hierarchy management, in ORM, 163
Highlighting chart data, 49–50
HMAC() function, for hash-based message authentication, 262
Hotfixes, managing from Administrator, 221
HQL (Hibernate Query Language)
extending SQL queries to ORM, 150
logging, 162
HTML5
accessing Google maps, 7–8
automatic fallback for browsers that do not support WebSocket, 96
browser support, 4
building REST application in, 91–92
Cold Fusion 10 support, 5
Geolocation API and <cfmap> tag, 5–7
video embedding support, 93–94
what’s new, 3–4
HTTP (Hypertext Transfer Protocol)
connecting to Exchange mailboxes, 281
cookie values used by session management, 261
invoking web services and, 57
overview of, 57
REST closely tied to, 77
standardization of communication, 56
as transport medium for SOAP requests, 68–69
HTTPS (Secure HTTP)
- connecting to Exchange mailboxes, 281
- securing cloud computing, 214
- securing web services, 67
Hybrid cloud, 203
Hypertext Transfer Protocol. see HTTP (Hypertext Transfer Protocol)

IaaS (Infrastructure as a Service), 202
IE (Internet Explorer)
- HTML5 support, 4
- video types supported, 94
IIS (Internet Information Services), 214
Implicit constructors, 121–122
Implicit indexing, in ORM, 158
Implicit notation, 123–124
index component
- methods, 143
- in place of CFINDEX tag, 141
- querying data source with, 142–143
Indexes, Apache Solr
- boosting custom fields and documents, 302–303
- DIH (DataImportHandler) and, 304–308
- filtering custom data, 300–302
- ordering results, 304
- specifying custom fields, 297–300
Indexes, ORM
- enabling entities for, 151–152
- of entities, 152
- explicit and implicit, 157–158
- purging, 158
Infrastructure as a Service (IaaS), 202
init(), initializing Java class constructors, 167
Inline (or inner) functions. see Closures
Installer, ColdFusion 10, 311
Instance types, EC2 (Elastic Cloud Compute), 206
Integration
- benefits of web services, 56
- copying Exchange folders, 288
- creating Exchange folders, 287
- as focus of ColdFusion, 55
- with Java. see Java integration
for Microsoft Exchange Server 2010, 281–283
for Microsoft Office documents, 295–296
modifying Exchange folders, 287–288
moving, deleting, or emptying Exchange folders, 289
overview of, 281
performing Exchange availability operations, 293–295
performing Exchange conversation actions, 292–293
retrieving Exchange conversations, 290–291
retrieving information about Exchange folders, 283–286
Interface, defining Java interface, 182
Internet Explorer (IE)
- HTML5 support, 4
- video types supported, 94
Internet Information Services (IIS), 214
invoke function
- approaches to CFC tag equivalents, 147
- invoking web service methods, 136
- JavaScript API function, 14
- for passing arguments, 134–135
- in place of <cfinvoke>, 133
- for reusing CFC instances, 135–136
- invoke method, CFProxy class, 177
invokeAndPublish function
- JavaScript API function, 14
- publishing messages, 31–32
Invoking web services, 56–57, 136
IP addresses, restricting access by, 221
IsSOAPRequest() function, 68–69
iTunes, playlist support, 100

.jar (Java archives)
- challenges of adding to loadPaths, 174–175
- loading Java classes, 168–169
Java
- disabling access to Java components, 252
- for-in loops applied to Java arrays, 130
- updates to embedded libraries, 319
Java Development Kit (JDK), 178–179
Java Enterprise Edition (JEE)
- ColdFusion running on JEE application server, 277
- deployment option, 311
Java integration
- accepting arguments, 180
- calling CFCs from Java, 175–176
- calling Java program with CFML page, 180–181
- CFProxy class and, 176–178
- challenges of dynamic loading of classes, 173–175
- compiling Java program, 178–179
- createDynamicProxy CFC function, 182–185
- defining application-specific class libraries, 169
- directInvoke argument of CFProxy class, 181
- DNS lookup example, 166–167
- invoking CFC methods, 179
- load path example, 170
- loading Java classes, 168–169
- looping over Java arrays, 185–186
- overview of, 165
- reloading changed classes, 172–173
reverse-DNS java library example of dynamic class loading, 171–172
this.javaSettings for application-specific class loading, 168
using Java libraries and classes, 167
Java Server Pages (JSPs), 168
Java Virtual Machine (JVM)
loading Java classes, 168
tuning, 277–279
javac command, compiling Java programs, 178–179
JavaScript, 102
API for setting source on media players, 95
HTML5 compatibility plug-ins, 4
messageHandler function, 12–13
playerOperation function, 102
WebSocket requiring, 10
what's new in HTML5, 4
working with JavaScript API, 13–16
JavaScript API function, 12–13
JavaScript Object Notation. see JSON (JavaScript Object Notation)
javaSettings property, application.cfc file, 169–170
JDK (Java Development Kit), 178–179
JEE (Java Enterprise Edition)
ColdFusion running on JEE application server, 277
deployment option, 311
jQuery, creating functions as arguments, 110
JRun Metrics logging, 317
JSON (JavaScript Object Notation)
refreshing charts, 51
SOAP vs. REST and, 75
specifying content types in REST, 82–83
styling charts, 42–43
XML compared with, 187
JSPs (Java Server Pages), 168
JVM (Java Virtual Machine)
loading Java classes, 168
tuning, 277–279
jvm.config file, 279, 312

K
Key:value pairs, support for colon separators in structures, 128–129
Kumar, Rupesh, 162

L
Layouts, templates for, 118–119
Libraries, Java
defining application-specific class libraries, 169
reverse-DNS java library example of dynamic class loading, 171–172
using, 167
Libraries, updates to embedded, 319
Licensing
changes to, 322
cloud computing and, 210–211
list method, in REST application, 89
listFilter example, inline functions in, 116
listLen function, handling form fields with same name, 120–121
Lists
inline functions and, 116
of scheduled tasks, 243–245
Load paths
challenges of adding .jar or .class files to loadPaths, 174–175
Java integration and, 170
Logging
access requests, 318
administrative changes, 219–220
enhancements to, 317–318
execution of scheduled tasks, 226
HQL logging, 162
Logon, limited to single administrator at a time, 315
Loop playback, video, 98–99
Looping
over Java arrays, 185–186
over queries, 129–130
Loose coupling, benefits of web services, 56
Lucene, updates to embedded libraries, 320

M
m3u playlist format, support in media players, 100
MathML (Mathematical Markup Language), 3
MAXLENGTH validation attribute, <cfparam>, 313
Measure service, features of cloud computing, 200
Media players
callback functions, 97–98
extending, 101
loop playback support, 98–99
playlist support, 100
poster image support, 99
setting source on, 95
style attribute for skinning, 97
title support, 99–100
type attribute for video support, 94–95
Memory allocation, JVM heap and, 278
Message handlers
JavaScript messageHandler function, 12–13
in news handler example, 20
Messages, WebSocket
filtering, 22
generating with CFCs, 31–32
JavaScript messageHandler function, 12–13
selectors, 25–27
subchannels, 23–25
Metadata
  applications and sessions, 313
  CFCs, 146

Methods
  CFCs, 146
  chaining method calls, 122–123
  collection component, 140
  index component, 143
  invoking CFC methods from Java, 179
  invoking web service methods, 136

Metrics logging, 317

Microdata, what’s new in HTML5, 4

Microsoft Excel, integration with ColdFusion, 295–296

Microsoft Exchange Server 2010
  creating folders, 287
  integration with ColdFusion, 281–283
  modifying folders, 287–288
  performing availability operations, 293–295
  performing conversation actions, 292–293
  retrieving additional folder information, 285–286
  retrieving basic folder information, 283–285
  retrieving child folder information, 286
  retrieving conversations, 290–291

Microsoft Internet Explorer. see IE (Internet Explorer)

Microsoft Office documents
  adding/maintaining data in Solr collections, 141
  integration improvements, 295–296

Microsoft Outlook. see Conversations, Microsoft Outlook

Microsoft Windows Media Player, 100

Microsoft Word, 295–296

MIME types
  checking MIME values of files, 260–261
  specifying for video, 94
  what’s new in HTML5, 4

Monitoring cloud, 215

Mozilla Firefox
  HTML5 support, 4
  video types supported, 94

MP4s
  playing using <cfmediaplayer>, 94–95
  playing using Adobe Flash, 97
  playing using HTML5, 96
  video types, 94

Multiserver deployments, architecture enhancements, 311–312

National Institute of Standards and Technology (NIST), 199

Network access, features of cloud computing, 200

Nielsen, Lasse, 107

NIST (National Institute of Standards and Technology), 199

Notation, implicit, 123–124

NULL values, in ColdFusion, 66

OAuth (Open Standard for authentication), 68

Object relational mapping. see ORM (object relational mapping)

Object-Oriented Programming (OOP), 60

Office documents. see Microsoft Office documents

Offline searches, ORM and, 155–157

Offline storage, what’s new in HTML5, 4

Ogg video type, 94

onAbort event
  added to Application.cfc file, 119–120
  <cfabort>, 313

On-demand self-service, features of cloud computing, 200

onError, callback function, 97–98

onException attribute, <cfschedule>, 237–238

onPause, callback function, 97–98

onWSAuthenticate method, WebSocket, 27–29

OOP (Object-Oriented Programming), 60

Open Source Media Framework (OSMF), 101

Open Standard for authentication (OAuth), 68

openConnection, JavaScript API function, 14

Opera
  HTML5 support, 4
  video types supported, 94

orderby attribute, <cfindex>, 304

ORM (object relational mapping)
  explicit and implicit indexing, 157–158
  hierarchy management, 163
  HQL logging, 162
  indexes, 151–152
  indexing ORM entities, 151–152
  overview of, 149
  purging indexes, 158
  searches, 149–150
  searching by level, 158–161
  searching multiple entity types, 154–155
  searching offline, 155–157
  searching ORM entities, 152–154
  searching relationships, 157

OSMF (Open Source Media Framework), 101
P

PaaS (Platform as a Service), 202
Parent-child relationships, in ORM hierarchy, 163

Passwords
  hash() function for securing, 262
  reset script, 220
  secure profiles, 252
PDFs, in Solr collections, 141
Performance improvements
  application-specific caching, 266–267
  built-in functions and, 269–271
  cache regions, 267–268
  caching and, 265–266
  EhCache version update, 275
  JVM tuning, 277–279
  new cache functions, 271–272
  overview of, 265
  query caching, 273–275
  Server Monitor support for caching, 275–277
Pie charts, 36–37, 41
Platform independence, web services and, 56
Player controls, video embedding and, 96–97
playerOperation, JavaScript, 102
Playlists, video embedding and, 100
plot attribute, <cfchart>, 49–50
Plotting multiple series, in charts, 44
Point-to-point applications, WebSocket, 32–34
POST operations, secure profiles, 252
Poster images, video embedding, 99
PowerPoint, integration with ColdFusion, 295–296
Previewing charts, 38–39
Prioritization, of scheduled tasks, 230–231
Private cloud, cloud deployment models, 203
Properties
  ORM searches at property level, 160–161
  Outlook conversations, 291
  Quartz scheduler, 250
Public cloud, cloud deployment models, 203
Publish, JavaScript API function, 14

Q

Quartz scheduler
  clustering tasks, 248
  Cron expressions, 235–237
  customizing, 250
  overview of, 226
Queries
  caching, 273–275, 320
  chart data, 38
  clearing query cache, 219
  of data sources, 142–143
  looping, 129–130
  script-based, 130
queryAddRow, improvements to CFML, 124–125
queryNew, improvements to CFML, 124–125

R

RDP (Remote Desktop Protocol), 207
RDS (Remote Development Services)
  administrative changes, 220
  sandbox permissions for, 315–316
  secure profiles, 252
Real Time Messaging Protocol (RTMP)
  extending media players, 101
  streaming video, 102–103
Recovery planning, cloud computing and, 214
Redundancy, cloud computing and, 214
Refreshing chart data
  with random chart type, 51
  in real time, 52–53
  updating entire chart, 50–51
RefreshWSDL, troubleshooting web services, 74
Registration, of REST services, 78–79
Relationships of ORM entities, searching, 157
RelaxNG type definitions, WSDL, 60
Reliability, advantages of cloud, 201
reloadOnChange, Java classes, 172–173
Remote Desktop Protocol (RDP), 207
Remote Development Services. see RDS (Remote Development Services)
renderer attribute, <cfchart>, 45
Rendering charts, 44–45
repeat attribute, <cfmediaplayer>, 98–99
repeat attribute, <cfschedule>, 234–235
Representational State Transfer. see REST (Representational State Transfer)
Resource pooling, cloud computing and, 200
Resources, securing, 251
REST (Representational State Transfer)
  Axis 2 support, 319
  building REST CFCs, 79–80
  cloud technologies and, 201
  customizing responses with, 92
  dynamically initializing REST services, 92–93
  registering REST services, 78–79
  sample application, 87–92
  SOAP compared with, 75–76
  specifying content types, 82–83
  testing REST CFCs, 80–82
  types of web services, 55
  what it is, 77–78
  working with subresources, 85–86
XML search filtering results of REST service call, 191
XML serialization and, 84–85
restInitApplication, REST services, 92–93
restSetResponse, REST services, 92
Reverse-DNS lookup
example of CFML/Java integration, 166–167
example of dynamic loading of Java classes, 171–172
Robust Exception Information, secure profiles, 252
Rooms, Microsoft Exchange Server
retrieving available, 295
viewing list of, 294
Root Path, REST services, 78
RPC Encoded, WSDL styles, 72
RTMP (Real Time Messaging Protocol)
streaming video, 102–103
runOnce attribute, <cfinclude>, 118–119, 312
sClosure() function, 117–118
Scripts. see <cfscript>
Scripts, cross-site, 252, 253–257
search component
basic search, 143–144
crawling web sites and, 145
in place of <cfsearch>, 143
Search engine optimization (SEO), 75
Searches. see <cfsearch>
Searches, Apache Solr
boosting custom fields and documents, 302–303
DIH (DataImportHandler) and, 304–308
filtering custom data, 300–302
ordering results, 304
specifying custom fields, 297–300
Security
backward-compatibility issues, 316–317
clickjacking and, 263
doT of cloud, 214
CSRF (cross-site request forgery) protection, 257–259
file upload protection, 260–261
Hash() function and, 262
HMAC() function and, 262
overview of, 251
sandbox permissions for RDS, 315–316
secure profiles, 252–253
Secure File Transfer Protocol (SFTP), 207
Secure HTTP. see HTTPS (Secure HTTP)
Secure profiles, 252–253
Secure Shell (SSH), 207
Secure Sockets Layer (SSL), 67, 313
Scheduling
adding and editing tasks, 247–248
application-level tasks, 226–228
chaining tasks, 231–233
changes to, 226
clustering scheduled tasks, 248–250
ColdFusion Administrator and, 247
comparing Standard and Enterprise editions, 225
Cron expressions for Quartz scheduler, 235–237
customizing Quartz scheduler, 250
deleting tasks, 230
excluding dates, 233–234
grouping tasks, 228–229
handling events, 239–243
handling exceptions, 237–238
handling misfires, 238–239
listing scheduled tasks, 243–245
logging task execution, 226
overview of, 225
pausing/resuming tasks, 246
prioritizing tasks, 230–231
Quartz engine for, 226
repeating tasks, 234–235
running tasks, 230
S3 (Simple Storage Service)
accessing with ColdFusion, 207–209
EC2 storage options, 206
overview of, 207
public cloud deployment, 203
service levels, 211–212
SaaS (Software as a Service), 202
Safari
HTML5 support, 4
video support, 94
Sandboxes
access rules, 251
RDS permissions, 315–316
secure profiles, 252
Scalable vector graphics (SVG), 3, 45
Searches.
see <cfsearch>
Searches, ORM
by level, 158–161
offline searches, 155–157
overview of, 149–150
searching entities, 152–154
searching multiple entities by type, 154–155
searching relationships, 157
SECURE attribute, <cfpop>, 313
Secure File Transfer Protocol (SFTP), 207
Secure HTTP. see HTTPS (Secure HTTP)
Secure profiles, 252–253
Secure Shell (SSH), 207
Secure Sockets Layer (SSL), 67, 313
Selenium
backward-compatibility issues, 316–317
clickjacking and, 263
of cloud, 214
CSRF (cross-site request forgery) protection, 257–259
file upload protection, 260–261
Hash() function and, 262
HMAC() function and, 262
overview of, 251
sandbox permissions for RDS, 315–316
secure profiles, 252–253
security WebSockets application, 27–30
Session feature improvements, 261
single sign-on authentication, 30–31
single user restriction in <cflogin>, 315
updates, 219
web services, 67–68
XSS (cross-site scripting) protection, 253–257
Security groups, EC2 (Elastic Cloud Compute), 207
Selectors, for filtering WebSocket messages, 25–27
SEO (search engine optimization), 75
Serialization, XML, 84–85
Server deployment option, 311
Server Monitor, 275–277
Server Settings, Scheduled Tasks link, 227
Servers
  adding to clustered scheduled tasks, 249
  managing updates, 221–224
  ORM searches at server level, 158
  securing, 252
  serverVersion attribute, 282–283
  WebSocket server-side functions, 21–22
Service level agreements (SLAs), 211
Service levels, cloud computing, 211–212
Service Mapping, REST services, 78
Service models, cloud computing, 201–202
Sessions
  security enhancements, 261
  session fixation protection, 316–317
  sessionGetMetaData() function, 313
Set as Default Application, REST services, 79
SetSource on media players, JavaScript API, 95
setting statement
  approaches to CFC tag equivalents, 147
  in place of <cfsetting>, 131–132
SFTP (Secure File Transfer Protocol), 207
showuser attribute, <cfmap>, 5, 312
Simple Storage Service. see S3 (Simple Storage Service)
Single sign-on authentication, 30–31
SLAs (service level agreements), 211
SOAP (Simple Object Access Protocol)
  Apache Axis implementation, 59, 319
  authorization by using SOAP headers, 68
  cloud technologies and, 201
  invoking web services and, 57
  overview of, 58
  platform independence and, 56
  requests, 68–70
  REST compared with, 75–76
  types of web services, 55
  web service engine options, 59–60
  WSDL support, 61
Software as a Service (SaaS), 202
Solr. see Apache Solr
Source code, of CFCs, 146
Special characters, in Cron expressions, 235–236
SQL (Structured Query Language)
  enabling SQL logging by ORM, 162
  searching databases with, 149–150
  secure profiles, 252
Square braces ([]), for use with arrays, 129
SSH (Secure Shell), 207
SSL (Secure Sockets Layer), 67, 313
Standard edition
  compared with Enterprise edition, 321
  scheduling in, 225
Statements, within scripts
  cookie statement in place of CFCOOKIE tag, 132–133
  setting statement in place of CFSETTING tag, 131–132
Statistics tab, Server Monitor, 275
Storage
  EC2 (Elastic Cloud Compute), 206
  temporary. see Caching
  what's new in HTML5, 4
Streaming video, 102–103
strict attribute, <cfinput>, 260
Strobe Media Playback
  DRM (digital rights management), 103
  extending media players, 101
  streaming video, 102–103
structFilter example, inline functions in, 116–117
Structs/structures
  colon separator in, 128–129
  inline functions and, 116–118
  serialization, 84
Structured Query Language. see SQL (Structured Query Language)
Subs
  event handling, 240
  web services, 63
Styles, chart, 42–44
Subresources, REST services, 85–86
SVG (scalable vector graphics), 3, 45
Syntax improvements, 118–121
System-level information, functions returning, 313

T

Tasks
  application-level vs. server level, 227–228
  scheduling. see Scheduling
TCO (total cost of ownership), cloud computing
  and, 201
TCPMon, troubleshooting web services, 75
Technologies, cloud computing, 200–201
Templates
  for layouts, 118–119
  secure profiles, 252
Themes, 217–218
This scope, CFCs (ColdFusion components), 123
  this.javaSettings, for application-specific class
  loading, 168
Time, dateTimeFormat() function, 313
Time to market, advantages of cloud, 201
Titles, video embedding and, 99–100
tokens, for setting attribute values, 49–50
Tomcat, as default server engine, 277
Total cost of ownership (TCO), cloud computing
  and, 201
Transactions, managing in cloud computing, 212
  Trusted cache, 219
  Tutorials, ColdFusion, 323
UDFs (user-defined functions)
building utilities with, 111
creating functions as arguments, 110–111
defining functions within existing functions, 108–109
example, 108
Uniform resource locators. see URLs (uniform resource locators)
 Universally unique IDs (UUIDs), 213, 252
Unix, password reset script, 220
update method, in REST application, 89
Updates
caching, 219
chart, 50–51
component, 121
security, 219
server, 221–224
URLs (uniform resource locators)
adding clickable URLs to charts, 45–48
benefits of using REST services, 77
for JSON and XML responses, 84–85
REST subresources and, 85–86
User-defined functions. see UDFs (user-defined functions)
Users, finding user availability for calendar events, 294–295
UUIDs (universally unique IDs), 213, 252
Variables, XPath, 192
Variables scope, CFCs (ColdFusion components), 123
Verity
deprecated support, 297
Verity-Solr migration, 310
VFS (virtual file system), 320–321
video element, HTML5, 93
Video embedding
callback events and error logs, 97–98
<cfmediaplayer>, 94–95
DRM (digital rights management), 103
enhancements to, 98
extending media players, 101
fallback plan for browsers not supporting, 95–96
HTML5 and, 93–94
loop playback support, 98–99
overview of, 93
player controls and, 96–97
playing YouTube video, 101–102
playlist support, 100
poster image support, 99
streaming, 102–103
title support, 99–100
what’s new in HTML5, 3, 5
Virtual file system (VFS), 320–321
Virtual Private Cloud (VPC), 210
Virtual private networks (VPNs), 210
Virtualization, cloud technologies, 200
VML, rendering charts and, 45
VPC (Virtual Private Cloud), 210
VPNs (virtual private networks), 210
W
W3C, Web Services Description Working Group, 58
WAR files, 312
watchExtensions, Java classes, 172–173
watchInterval, Java classes, 172–173
WDDX, XML style, 84
WDSL, Axis 2 support, 319
Web browsers
fallback plan for browsers not supporting video embedding, 95–96
HTML5 support, 4
video support, 94
WebSocket support, 10, 34
Web pages, in Solr collections, 141
Web servers, securing cloud computing, 214
Web service engines
choosing, 71
options, 59–60
specifying, 71–72
Web Service Security (WSS), 68
Web services
access control, 67–68
application-level settings, 70
architecture of, 57–58
best practices, 73–74
building, 60–62
cloud technologies, 201
comparing SOAP with REST, 75–76
complex data types, 63–64
configuring, 73
consumer, 62–63
including ColdFusion types in WSDL, 71
invoking, 56–57
invoking methods, 136
overview of, 55
passing arguments to, 64–66
securing, 67
service engine options, 59–60
specifying service engines for, 71–72
stubs, 63
troubleshooting, 74–75
updates to embedded libraries, 319
what they are, 55–56
working with multiple arguments, 66
working with SOAP requests, 68–70
WSDL styles, 72–73
Web Services Description Language. see WSDL

Web Services Description Working Group, W3C, 58
WebDAV, 281–282
WebM, 94
WebSocket
  browser support, 10
  `<cfwebsocket>`, 11–12
  channel definitions, 11
  ColdFusion Administrator options, 34–35
  error handling and unsupported browsers, 34
  filtering messages, 22
  generating messages with CFCs, 31–32
  JavaScript `messageHandler` function, 12–13
  point-to-point applications, 32–34
  secure profiles, 252
  security groups for EC2, 207
  selectors, 25–27
  server-side functions, 21–22
  single sign-on authentication, 30–31
  subchannels, 23–25
  what is it, 9–10
  what's new in HTML5, 4–5
  working with CFC (ColdFusion component) handlers, 16–21
  working with JavaScript API, 13–16
WHERE clause, SQL, 149–150
Windows Media Player, 100
Windows OSs
  AMIs (Amazon Machine Images), 215
  password reset script, 220
Word, Microsoft, 295–296
wschannels, defining WebSocket channels, 10
WSDL (Web Services Description Language)
  building web services, 61
  choosing and specifying web service engines, 71–72
  choosing styles, 72–73
  including ColdFusion types in, 71
  invoking web services and, 57
  overview of, 58
  platform independence and, 56
  refreshing stubs, 63
  SOAP vs. REST and, 75
  troubleshooting web services, 74
  web service engine options, 60
wsGetAllChannels function, WebSocket, 21–22
wsGetSubscribers function, WebSocket, 21–22
wsPublish function, WebSocket, 17–18
WSS (Web Service Security), 68
wsversion, troubleshooting web services, 74

X

XML (Extensible Markup Language)
  enhancements to, 187
  mapping to XPath variables, 192
  overview of, 57
  REST support for XML serialization, 84–85
  SOAP vs. REST and, 75
  specifying content types in REST, 82
  standardization of communication, 56
  updates to embedded libraries, 319
  WSDL support, 60
  XPath and, 187–191
  XSLT and, 192–195
XPath
  benefits of XPath searches, 189–191
  comparing versions of, 189
  examples, 187–188
  mapping to XPath variables, 192
XSLT (Extensible Stylesheet Language Transformations)
  examples, 192–194
  new features in version 2.0, 194–195
  overview of, 192
XSS (cross-site scripting) protection, 252, 253–257
X-Y charts, 41

Y

YouTube video, 101–102

Z

ZingChart
  as charting engine, 40
  Cold Fusion Builder and, 42–44
  exposing events from, 45
  highlighting important data in, 49–50
  updating `<cfchart>`, 37
  updating entire chart, 50–51
Zooming charts, 38–39