Adobe

ADOBE COLDFUSION 10

ColdFusion 10 Enhancements and Improvements

web application construction kit

Ben Forta

with Charlie Arehart , Rob Brooks-Bilson, Raymond Camden, Ken Fricklas, Hemant Khandelwal, and Chandan Kumar

Adobe

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

Adobe® ColdFusion® 10
Web Application Construction Kit:
ColdFusion® 10 Enhancements and Improvements
Copyright © 2013 by Ben Forta

Adobe Press Editor: Victor Gavenda
Project Editor: Nancy Peterson
Production Editor: Tracey Croom
Development Editor: Judy Ziajka
Proofreaders: Liz Welch and Scout Festa
Technical Editor: Hemant Khandelwal
Compositor: Danielle Foster
Indexer: Jack Lewis
Cover Designer: Charlene Charles-Will
Cover Compositor: Mike Tanamachi

Adobe Press books are published by Peachpit, a division of Pearson Education located in San Francisco, California. For the latest information about Adobe Press books, go to www.adobepress.com. To report errors, please send a note to errata@peachpit.com. For information about getting permission for reprints and excerpts, contact permissions@peachpit.com.

Printed and bound in the United States of America

ISBN-13: 978-0-321-89096-2
ISBN-10: 0-321-89096-5

9 8 7 6 5 4 3 2 1

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

This page intentionally left blank

CONTENTS

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 Ki*t (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 1—*Adobe ColdFusion 9 Web Application Construction Kit, Volume 1: Getting Started* (ISBN 0-321-66034-X)** contains Chapters 1–21 and is targeted at beginning ColdFusion developers.
- **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.

CHAPTER 4 Web Services

IN THIS CHAPTER

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)–complaint 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).

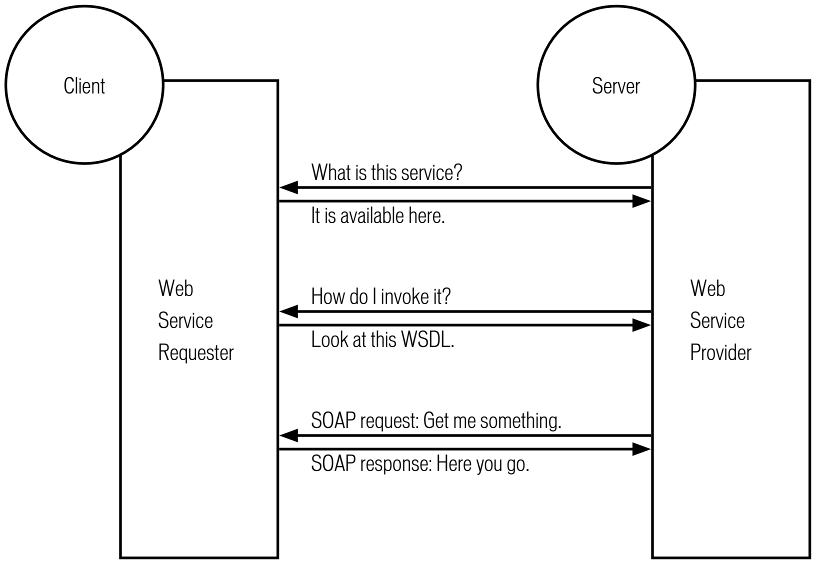


Figure 4.1
Typical web service invocation.

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.

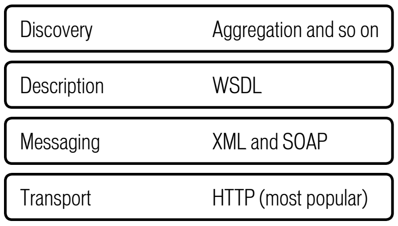


Figure 4.2
Web services architecture.

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 `?wsdl` 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/wsdl.

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.

Listing 4.1 /cfwack/4/hello.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

```
<!— 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.

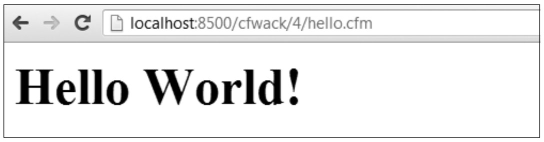
Listing 4.3 Calling a Web Service with `<cfinvoke>`

```
<cfset wsURL = "http://localhost:8500/cfwack/4/hello.cfc?wsdl">
<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.



localhost:8500/cfwack/4/hello.cfm

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>`

```
<cfset wsURL = "http://localhost:8500/cfwack/4/hello.cfc?wsdl">
<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>
  wsURL = "http://localhost:8500/cfwack/4/hello.cfc?wsdl";
  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

```
<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"/>

        <!---
```

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).

Listing 4.7 /cfwack/4/complex.cfm

```
<cfset wsURL = "http://localhost:8500/cfwack/4/complex.cfc?wsdl">

<cfset varStruct = {key1:"value 1", key2:"value 2"} >
<!-- Passing arguments with cfinvokeparam --->
<cfinvoke webservice = "#wsURL#"
          method = "echoStruct"
          returnVariable = "result">
   <cfinvokeargument name="argStruct" value="#varStruct#" >
</cfinvoke>

<h2> Dumping struct </h2>
<cfdump var="#result#"/>

<cfset varQuery = QueryNew("column1,column2,column3") >
<cfset QueryAddRow(varQuery,["row 1", "row 2", "row 3"])>

<!-- Passing arguments inline as key value pair --->
<cfinvoke webservice = "#wsURL#"
          method = "echoQuery"
          argQuery = "#varQuery#"
          returnVariable = "result">
</cfinvoke>

<h2> Dumping query </h2>
<cfdump var="#result#"/>

<!-- Passing arguments as argument collection --->
<cfinvoke webservice = "#wsURL#"
          method = "echoAny"
          argumentcollection = "#{argAny:'passing a string'}#"
          returnVariable = "result">
</cfinvoke>

<h2> Dumping String </h2>
<cfdump var="#result#"/>
```

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).

Listing 4.8 /cfwack/4/secure/Application.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>
```

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

```
<cfset wsURL = "http://localhost:1234/cfwack/4/secure/hello.cfc?wsdl">

<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 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>

        <cfreturn result>
    </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>
 wsURL = "http://localhost:8500/cfwack/4/soap.cfc?wsdl";
 ws = CreateObject("webservice", wsURL);

 // Set the username and passwordheader as a string.
 addSOAPRequestHeader(ws, "http://somenamespace/", "username", "user");
 addSOAPRequestHeader(ws, "http://somenamespace/", "password", "pass");
```

Listing 4.11 (continued)

```
     // 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");
    </cfscript>

    <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 headers, `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 services 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

```
<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>
```

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:

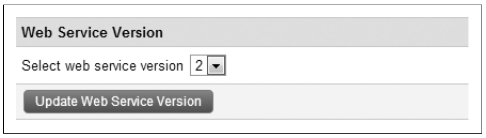
- Web Service Engine Version 1: Use this version if you want to publish WSDL in RPC style, or if you do not want all your existing web service clients to refresh their generated stubs. You should also use this version when you have web service clients on ColdFusion 9 and you use complex data types.
-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


Web Service Version

Select web service version 2

Figure 4.4
Changing the web service version in ColdFusion Administrator.

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 `XMLHTTPRequests` (XHR) , and SOAP benefits web services accessed by an intermediate server or middleware.

This page intentionally left blank

This page intentionally left blank

INDEX