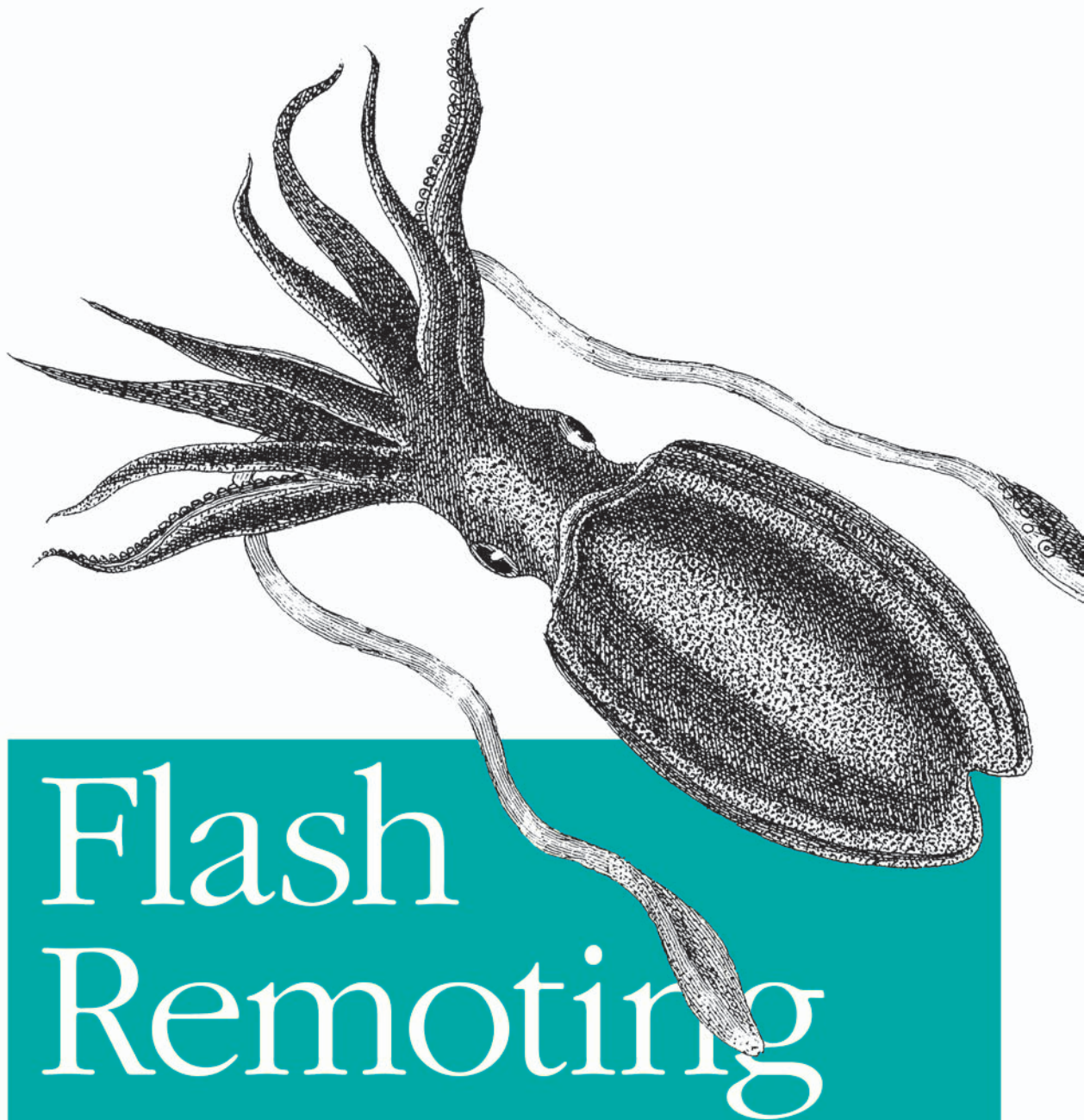


Connecting Flash Applications to Remote Services



Flash Remoting

The Definitive Guide

O'REILLY®

Tom Muck

Flash Remoting: The Definitive Guide



Flash Remoting lets developers easily integrate rich Macromedia Flash content with applications that are built using Macromedia ColdFusion MX, Microsoft .NET, Java, PHP, or SOAP-based web services. The result is complex client/server applications that more closely resemble desktop applications than click-and-wait HTML pages. Flash Remoting handles the communication with the application server to create fast and flexible web applications with a Flash front end.

The potential uses for Flash Remoting are practically endless. *Flash Remoting: The Definitive Guide* helps you understand this breakthrough technology and use it to build your own Rich Internet Applications (RIAs) that connect to remote databases and services.

The book begins with Flash Remoting basics: setup, installation, and an introduction to its underlying concepts. Then, you'll gain insights into Flash Remoting internals, recordsets, and UI components. The book is full of examples in multiple languages; individual chapters cover Flash Remoting development with ColdFusion, Server-Side ActionScript, Java, ASP.NET, and PHP. The last section covers more advanced Flash Remoting techniques, such as calling web services from Flash Remoting, extending objects and UI controls, best practices, and debugging. Plus, there is a detailed chapter demonstrating a real-world application. The book concludes with a Flash Remoting API reference.

This book shows you how to:

- Design a Flash Remoting application with a multitiered architecture that separates presentation and business logic from the data tier
- Implement efficient front-end interfaces to a variety of back-end application servers
- Access remote databases using ColdFusion, Java, ASP.NET, PHP, or Server-Side ActionScript
- Access existing SOAP-based web services from Flash
- Implement paging, caching, and security mechanisms to create efficient and secure applications
- Create full-fledged applications using data-aware Flash components
- Design and implement your Flash Remoting application using best practices
- Analyze Flash Remoting communications using the NetConnection debugger

Developers who are looking to create Rich Internet Applications with Flash will find *Flash Remoting: The Definitive Guide* indispensable.

www.oreilly.com

US \$39.95

CAN \$61.95

ISBN-10: 0-596-00401-X

ISBN-13: 978-0-596-00401-9



Flash Remoting

The Definitive Guide

Other resources from O'Reilly

Related titles Programming ColdFusion MX ActionScript for Flash MX:
ActionScript for Flash MX The Definitive Guide
Pocket Reference ActionScript Cookbook

oreilly.com *oreilly.com* is more than a complete catalog of O'Reilly books. You'll also find links to news, events, articles, weblogs, sample chapters, and code examples.



oreillynet.com is the essential portal for developers interested in open and emerging technologies, including new platforms, programming languages, and operating systems.

Conferences O'Reilly & Associates brings diverse innovators together to nurture the ideas that spark revolutionary industries. We specialize in documenting the latest tools and systems, translating the innovator's knowledge into useful skills for those in the trenches. Visit *conferences.oreilly.com* for our upcoming events.



Safari Bookshelf (*safari.oreilly.com*) is the premier online reference library for programmers and IT professionals. Conduct searches across more than 1,000 books. Subscribers can zero in on answers to time-critical questions in a matter of seconds. Read the books on your Bookshelf from cover to cover or simply flip to the page you need. Try it today with a free trial.

Flash Remoting

The Definitive Guide

Tom Muck

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

Flash Remoting: The Definitive Guide

by Tom Muck

Copyright © 2003 O'Reilly & Associates, Inc. All rights reserved.
Printed in the United States of America.

Published by O'Reilly & Associates, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly & Associates books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*safari.oreilly.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Editor: Bruce Epstein
Production Editor: Genevieve d'Entremont
Cover Designer: Emma Colby
Interior Designer: David Futato

Printing History:

September 2003: First Edition.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly & Associates, Inc. Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly & Associates, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps. The association between the image of a cuttlefish and the topic of Flash Remoting is a trademark of O'Reilly & Associates, Inc.

While every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

ISBN: 0-596-00401-X

[C]

*Dedicated to the memory of my father-in-law,
Stanley S. Lee . . . a truly original man who
succeeded in living life on his own terms.
Dad, your character is an inspiration. It was a
privilege to have known you and I will miss you.*

—Tom Muck

Table of Contents

Preface	xiii
----------------------	-------------

Part I. Remoting Fundamentals

1. Introduction to Flash Remoting	3
What Is Flash Remoting?	3
How Does Flash Remoting Work?	5
Benefits	10
Flash Remoting Requirements	15
Supported Platforms	17
Hello World	19
Workflow Example	32
Architecture for Flash Remoting Applications	33
2. Installing, Configuring, and Using Flash Remoting	39
Installing Flash Remoting	39
Naming Your Services	51
The Authoring Environment	55
Sequence of Events in Flash	67
Making a Remote Call	69
3. Client/Server Interaction, UI Components, and RecordSets	79
User Interface Components	79
Flash UI Components Set 2 and Flash Charting Controls	83
Macromedia's DRK	85
Text Objects	87
Building Forms in Flash MX	90

The RecordSet Class	92
DataGlue	105
4. Flash Remoting Internals	113
The NetConnection Class	114
Using NetServices	114
Creating Responder Functions	120
RecordSet Object	129
The Timeline	142
Catching Errors	143
Registering Objects	147
<hr/>	
Part II. The Server-Side Languages	
5. Flash Remoting and ColdFusion MX	161
Introduction to ColdFusion MX	161
How ColdFusion Fits into Flash Applications	162
Service Name Mappings	166
ColdFusion Pages Versus ColdFusion Components	186
Loading Query Data Incrementally	187
ColdFusion Security: Authenticating Users	194
Updating Data on the Server	200
6. Server-Side ActionScript	217
What Is Server-Side ActionScript?	217
The CF Object	219
When to Use Server-Side ActionScript	229
Datatype Conversions	229
Handling Errors with try/catch	230
Extending Server-Side ActionScript with Java	233
7. Flash Remoting and Java	249
The Flash Remoting Gateway	249
Supported Platforms	250
Setting Up Flash Remoting for J2EE	250
Service Types	252
Datatype Conversions	257
Service Lookup	266

Invoking Service Methods	269
Architecture and Design	272
Referencing the Request and Session	276
Saving and Sharing State	279
Databinding	283
Security	285
OpenAMF	292
8. Flash Remoting and .NET	293
Overview of .NET	293
Connecting to .NET	294
Datatype Conversions	302
Database Access with ADO.NET	310
State Management	318
Error Handling in Flash Remoting with .NET	321
9. Flash Remoting and PHP	324
Introduction to PHP	324
Installing AMFPHP	326
Datatype Conversions	328
Using AMFPHP with Web Services	329
Using AMFPHP with PHP Classes	330
Working with a Database	333

Part III. Advanced Flash Remoting

10. Calling Web Services from Flash Remoting	343
What Is a Web Service?	343
Why Use Flash Remoting?	348
Web Services from a ColdFusion Server	352
Web Services from an ASP.NET Server	353
Wrapping J2EE and JRun Web Services	357
Web Services from PHP	359
BabelFish Web Service	359
Datatype Conversions	362
Creating a Flash MX Web Service Extension	369

11. Extending Objects and UI Controls	373
Principles Behind UI Components	373
DataProviderClass and DataGlue	374
Enhancing a Standard Control	378
Enhancing the RecordSet Class for Interactivity	387
Adding Validation to a TextField	396
12. Flash Remoting Best Practices	399
Separation of Tasks	399
Separation of Functionality	401
Server-Proofing the Application	403
Component Use	404
Clean API	407
Loops and Repeated Operations	408
OOP or Not OOP	411
ColdFusion RecordSets	427
Implementing Caching	427
What to Walk Away With	429
13. Testing and Debugging	430
Debugging Flash Remoting Applications	430
Trapping Errors	435
The Flash ActionScript Debugger	437
NetConnection Debugger	443
NetDebug.as	449
Trace Debugging	450
Testing Server-Side Code	452
Using a TCP Trace Utility	460
Debugging SQL Code	464
14. Real-World Application	467
General Considerations	467
Requirements	468
Specifications	468
The Modules	470
Flash Remoting Code	500
Testing and Debugging	522
Completed Application	523
Future Expansion	523

15. Flash Remoting API	525
------------------------------	-----

Part IV. Appendixes

A. ActionScript Datatype Conversion	575
B. Books and Online Resources	582
C. Specification and Implementation for a Real-World Application	589
Index	593

Preface

The Web has always been an expanding, changing medium. What was in vogue just a few years ago might be completely abandoned today, and what is in vogue today may be completely abandoned tomorrow. Java™ applets were considered a breakthrough technology that was going to revolutionize the Web when they first came out. Java applets failed to take hold for several reasons, chief among them the complexity of the Java language for nonprogrammers, the download size for anything more than a simple applet, and the security restrictions that effectively tied the hands of the applet. ASP was Microsoft's attempt to replace Java applets with server-side applications. JSP learned from the mistakes of ASP and provided a much more robust solution by offering compiled pages rather than interpreted pages. ASP.NET, in turn, learned from the mistakes of JSP by providing a framework that can be accessible to a variety of different languages using a common runtime environment.

And Flash has learned from the mistakes of those early Java applets. The Flash browser plugin is small, self-contained, and ubiquitous. Flash is attractive to designers as well as programmers, and it provides rich functionality that enhances the end user's experience. Also, a Flash movie looks essentially the same on a Windows machine as it does on a Macintosh.

Some things have not changed, however. The Web is still primarily based in HTML. Even though XHTML is coming into prominence, it is just a syntactically standardized version of HTML that conforms to XML specifications. Flash solves many of the deficiencies of HTML by improving on animation limitations of DHTML, allowing for true interactivity, offering greater cross-platform and cross-browser consistency, and allowing upgrades of the Flash Player without requiring the user to upgrade his browser. By making Flash consistent across platforms, Macromedia has ensured that the Flash designer can virtually eliminate the laborious trial-and-error work that HTML web page designers go through to keep their pages consistent across platforms and browser types.

Macromedia Flash Remoting MX, or simply *Flash Remoting*, first offered with Flash MX, allows sophisticated datatypes to be passed from the server to the client and back without the speed limitations, bulk, and manual serialization/deserialization required by the XML techniques in previous versions of Flash.

In a traditional HTML application, a typical user experience is like this:

1. The user comes to your site. He enters something into a search field and clicks the Submit button.
2. The user waits for a response from the server. His browser loads the page, allowing the user to view the search results.
3. If the results span more than one page and the user clicks the link for the next set of results, the search request is sent to the server again. The server conducts the search again and sends back the second page of results.
4. Again, a new page is loaded into the browser.

You can liken this experience to a very inefficient phone conversation with a friend. You ask a question and hang up the phone. Your friend calls back and answers the question, then hangs up. You call him back, add something to the conversation, and hang up again. Two tin cans with a taught string between them gives you a more advanced communication method. Yet this is similar to how the client/server communication with a browser operates, and it is the prevailing standard on the Web.

HTML can build primitive web applications, but its main advantage is its ubiquity. However, Flash Player 6, which plays back content authored in Flash MX (including Flash Remoting applications) has over 85% penetration and allows developers to deploy much more sophisticated web applications. Flash Player 7 is in public beta, and by the time you read this, the release version will be available. Flash Remoting offers seamless communication between the client browser and the remote server. Each revision of the Flash Player, whose uptake is much faster than browser updates, essentially upgrades the Web. Only minor changes have been made to Flash Remoting in Flash MX 2004 and Flash MX Professional 2004 (Flash Pro), although the latter offers enhanced support for SOAP-based web services. Regardless of whether you are using Flash MX, Flash MX 2004, or Flash Pro, the principles and examples shown throughout this book apply equally.

Singing the praises of Flash Remoting is not to say that HTML does not have its place—it clearly does—but Flash offers user interactivity closer to a desktop application than a traditional HTML web site. Also, because ActionScript's syntax is nearly identical to JavaScript, web scripters can easily transition to Flash. Flash MX also introduced Flash UI components, which are configurable and skinnable interface elements, such as checkboxes and list boxes. In previous versions of Flash, UI functionality had to be created for each new application. Components allow developers to concentrate on the application's functionality rather than worry about mundane aspects of buttons and form elements. Components share a common API that

leverages easy-to-use methods. This brings rapid application development (RAD), familiar to Delphi and Visual Basic users, to Flash-based web applications. Components are also extremely flexible, allowing designers to skin the components to fit seamlessly with the site design.

Drumbeat 2000, a web development program from Elemental Software (later purchased by Macromedia), tried to abstract HTML into a visual interface by turning HTML elements into JavaScript objects. Strongly resembling Visual Basic, the interface of Drumbeat 2000 used drag-and-drop components that were thinly disguised HTML tags and server-side script objects. Unfortunately, HTML was not intended for a point-and-click environment or an event-based model. Drumbeat 2000 died a quick death because the HTML/JavaScript objects slowed down an already slow technology, although parts of it were rolled into UltraDev and Dreamweaver MX.

Flash, on the other hand, is designed around an event-based GUI that thrives on point-and-click interaction with efficient self-contained ActionScript objects. Flash allows designers to customize the user interface, limited only by their imagination. A Flash designer can utilize standard prebuilt interface objects or build his own to create the interaction between his application and the end user. The ActionScript programmer can use these components to call server-side methods through Flash Remoting. This allows the Flash movie to do what it does best—interact with the user—and allows the server-side application to do what it does best—process information.

Flash Remoting technology adds true client/server communication to browser-based applications because it is not page-based; it is based on a single interface that loads once for the entire application. The Flash movie creates a one-time connection to the server. The *application state* is maintained within the Flash Player. Gone is the click/wait/reload approach of HTML. As of Flash MX, you can build your web application as a unit with Flash as the front end and your application server on the back end. The communication with the server is handled by Flash behind the scenes. When you build an application with Flash Remoting, the user experience is similar to what you would expect from a desktop application.

Take, for example, an interaction with an online store. A typical HTML page can consist of dozens of files and be 20–100 KB or more. But what about subsequent pages? Even if a page is remarkably similar to the one that came before, each page is often loaded in its entirety from the server (unless that page is already cached locally, in which case it still has to be rerendered by the browser). Menus, headings, copyright notices, and so on might be identical on every page, yet they are reloaded each time. There has to be a more efficient means of downloading and rendering page content.

Enter the Flash Player and Flash Remoting. With Flash Remoting, the browser needs to download only the subset of information that has changed from page to page, which is typically fed from the application server. When you click a Search button in

a Flash application that uses Flash Remoting, the request is sent to the server in a small ActionScript Message Format (AMF) packet, and another AMF packet is sent back to the browser. The Flash movie loads the information contained in the packet, such as a recordset, array, or other datatype, and utilizes it as needed.

If successive screens are highly similar, instead of loading each 25 KB page separately, Flash needs to download only the portion that differs—the information to be displayed instead of existing interface elements. If data is required, the AMF format ensures it can be downloaded efficiently in a mere few hundred bytes. There is no need to completely redraw the entire page as a new page loads in, because Flash Remoting is supplying only data from the server and the Flash movie uses the data as needed. Also, Flash can often download data in the background so it is ready when needed. In most cases, the user isn't even aware that the server was contacted.

You can picture poor old HTML just sitting there thinking, “now why didn't I think of that?” Developers are often slow to adapt, however. Many HTML developers are unwilling to admit that HTML, while apt for simple pages, was never designed for complex interactivity, animation, and sound. As such, it is not surprising that Flash is much more convenient for creating interactive sites. Nevertheless, Flash and Flash Remoting are bringing the same interactivity and usability to the Web that GUIs brought to the desktop a generation ago.

Remoting: More Than Just a Name

Flash Remoting is a technology that resides on the remote application server—hence the name. It acts as a *gateway* to the server and translates client calls from a Flash movie into calls that the server can understand. It is included with ColdFusion MX Server and JRun 4 and offered as an add-on for ASP.NET and J2EE-compatible servers, such as IBM WebSphere. Third-party developers have built versions for PHP and Perl as well.

Prior versions of Flash were able to communicate with a server in a limited fashion using name/value variable pairs or XML. Flash Remoting offers the following advantages over these methods:

- A Flash movie, through the server-side Flash Remoting gateway, can call server-side methods directly, not merely pass values or files back and forth.
- The client- and server-side XML or name/value parsing required for Flash 5 interaction is virtually eliminated with Flash Remoting, although it can still be used effectively in some situations.
- Because the server-side Flash Remoting gateway can communicate directly with the Flash movie, complex objects, such as arrays, structures, and recordsets, can be transferred without serialization/deserialization.

Also, because the server contains functionality that can be accessed directly (such as calling a server-side method by name from within the Flash movie), the Flash code can be more concise. Recordsets can be loaded into Flash movies using a *RecordSet* object, making it easy to sort and page through results directly from the client browser. In addition, the *DataGlue* class (part of the client components of Flash Remoting) offers a set of functions that allows you to bind Flash user interface elements to the data. Flash Remoting also offers debugging of client-side code and server communication using the *NetConnection debugger* and a *service browser* that allows you to introspect remote services from the Flash authoring environment. With all of these features, web programming has finally come of age.



Throughout this book we refer to *recordsets* (data retrieved from a database on a server) and client-side *RecordSets* (the ActionScript class that is used in Flash Remoting). A third term—*resultset*—is used when talking about the *java.sql.resultset* class in Java.

Flash Remoting makes it a snap to connect to ColdFusion, J2EE, ASP.NET, and PHP applications and the back-end databases that these server technologies support, such as SQL Server, Oracle, DB2, Access, PostgreSQL, and MySQL. Using Flash Remoting, you can create sophisticated applications with a clear line drawn between the client-side interface (Flash) and the server-side logic (business logic). The Flash application can be deployed in the browser or on the desktop. Also, the server-side logic need not be tied to a Flash interface: your server-side code does not require any Flash-specific syntax. In addition to servicing the Flash client, the server-side methods of the application can be utilized by other server-side pages supplying HTML content, or even a .NET desktop application.

Flash Remoting, in conjunction with ColdFusion MX, JRun, and other technologies, also replaces much of the functionality of Macromedia's extinct Generator server-side Flash technology. Generator templates are no longer supported, but Macromedia offers a transition path to Flash Remoting. For more information on the transition, go to:

<http://www.macromedia.com/software/generator/productinfo/faq/>

What Is Remoting Used For?

I've talked about the benefits of Flash Remoting, but what can you build with it? Here are some of the possible uses of Flash Remoting:

- Any application that requires a connection to a database, filesystem, or other server-side technology.
- Online stores that feature catalogs and shopping cart systems. The entire user experience can be consolidated into one central interface with no page reloads.

- Sound and video clip libraries. Using the streaming audio and video capabilities of Flash in conjunction with Flash Communication Server (FlashCom), along with the server-side capabilities of an application server, you can offer searchable, browsable clip libraries.
- Banner ads with built-in shopping carts, click-through tracking, and full-site search capabilities.
- New controls that can be used in place of HTML elements that leverage the server-side capabilities of ColdFusion, ASP.NET, or J2EE.
- Extensions to Flash, Dreamweaver, Fireworks, or other applications that provide instant access to a remote support site or reference material.
- An online auction interface that stores your own watched items locally and polls the remote web service for changes in bids.
- Front-ends to databases for administrators. You can give your client an administrative interface so that she can access tables and raw data without giving her the keys to the store, so to speak.

The architecture of Flash Remoting makes it easy for Flash programmers to develop Flash movies, knowing only the remote method names and functionality. They can access server-side functionality using familiar ActionScript syntax. Likewise, server-side coders can build functionality on the server without knowing how the Flash movie is going to interact with it. They merely have to provide an interface to the method and return a result. That result can be virtually anything, such as a record-set, string, number, array, or structure. Complex applications can be broken down into component parts, with designers building the Flash interfaces, ActionScript programmers coding the client-side Flash Remoting code, and server-side programmers (ColdFusion, PHP, Perl, Java, or ASP.NET) supplying the server-side functionality.

What This Book Contains

Flash development encompasses drawing, programming, animating, and working with sound, images, and video. You can be proficient at one or many of these technologies. There are numerous books that cover Flash, each with its own merits and approach. Knowledge of ActionScript is essential to understanding Flash Remoting. (However, the ActionScript gymnastics that Flash 5 developers had to employ to communicate with a server are virtually eliminated with Flash Remoting.)

This book teaches Flash Remoting for programmers, because it is a technology for programmers. I assume you are one of the following:

- A Flash (ActionScript) programmer who wants to use Flash Remoting to communicate with a server
- A server-side application developer who wants to expand his knowledge to Flash development

I discuss Flash Remoting using terminology that an experienced programmer will appreciate, but I try to keep the discussion within the reach of aspiring programmers as well. I also explain the best practices for working with Flash Remoting. Although best practices are often subjective, I try to be as objective as possible or offer consensus conclusions about best practices.

What This Book Is Not

Flash is traditionally an environment for designers, but the design aspects of Flash are not covered in this book. This book assumes you are familiar with the Flash authoring environment. When I mention the Actions panel, for example, I assume you know to find it under Window → Actions (F9). If not, go through the Flash tutorials under Help → Flash Tutorials, or read an overall tutorial book on the Flash MX environment and workflow, such as *Macromedia Flash MX Hands-On Training* (Peachpit Press).

Flash Remoting is not complex, but it helps to know the Flash authoring tool. If you are familiar with other programming environments, such as the Visual Basic or Delphi environments, learning the Flash interface is easy. Coming from a Visual Basic background, I felt right at home in Flash MX. Flash Pro extends the forms-based visual metaphor, making it even more accessible to Visual Basic developers.

This book assumes you know basic programming concepts—such as loops, variables, and arrays—and ActionScript programming techniques in particular. If you don't know ActionScript but are proficient in another language, such as JavaScript, C++, Java, or CFML, see *ActionScript for Flash MX: The Definitive Guide* by Colin Mook (O'Reilly). That book is recommended for ActionScript programmers of all levels, including those wanting to learn ActionScript (it too assumes familiarity with the Flash authoring environment's GUI).

Object-oriented programming (OOP) will play an important role in this book, but again, this isn't a book on OOP. I assume you have at least a passing knowledge of OOP; the book uses OOP techniques when they make sense, but it is not bound by them. There are some OOP purists who may scoff at some of the techniques, but it is my goal to show you the techniques you need to utilize Flash Remoting. OOP is, after all, only a methodology. It is most often the best tool for the job, but I don't want to burden the code with OOP for its own sake.

This book does not teach ColdFusion, C#, or Java; however, basic programming knowledge is sufficient to understand the examples even if you don't know much about server-side programming. Because Flash Remoting can be used in a variety of environments, examples are presented in different technologies. Wherever applicable, the server-side examples show ColdFusion (CFML), J2EE (Java), PHP, and ASP.NET (C# or VB) code. The Flash ActionScript, in most cases, remains essentially the same. Independence of the server-side server model is another cool benefit of Flash

Remoting. O'Reilly has a slew of excellent books on C#, Java, PHP, and ASP.NET. See Appendix B for some recommendations, or see the O'Reilly catalog page at <http://www.oreilly.com/catalog> for a more complete list. For guidance on ColdFusion, see *Programming ColdFusion MX* by Rob Brooks-Bilson (O'Reilly).

The Book at a Glance

This is the first edition of *Flash Remoting: The Definitive Guide*. It covers new ground because Flash Remoting is a relatively new technology. Most of the examples in the book centralize the code on the first frame of the timeline or in an external include file.

Flash Remoting works with several different server-side technologies. You should skim the sections that cover alternative server models, as they contain useful nuggets applicable to other platforms. Each chapter may be read individually or as parts of a whole. The first three chapters will be crucial to understanding Flash Remoting and should be read sequentially.

The chapter breakdown is as follows.

Part I: Remoting Fundamentals

Part I covers the setup and installation of Flash Remoting and offers a basic introduction to its use. It also offers details on core concepts and Remoting internals.

Chapter 1, *Introduction to Flash Remoting*

A general introduction to Flash Remoting and related technologies. It implements a sample “Hello World” application for all supported platforms, which demonstrates the simplicity of the technology.

Chapter 2, *Installing, Configuring, and Using Flash Remoting*

Gives a complete rundown of what is needed to implement Flash Remoting and describes the basic installation processes for each component. The Flash authoring interface is described as it relates to Flash Remoting.

Chapter 3, *Client/Server Interaction, UI Components, and RecordSets*

The Flash MX UI components are covered as they relate to Flash Remoting, along with several components released as commercial add-ons by Macromedia. In addition, techniques to build forms in Flash MX are described. Examples highlight each key point.

Chapter 4, *Flash Remoting Internals*

Explains the Flash Remoting API by dissecting the various classes that are installed when you install the Flash Remoting components and methods of those classes. Again, each important point is shown using examples that you can run on your own system.

Part II: The Server-Side Languages

Part II covers the server-side environments that Flash can communicate with via Flash Remoting.

Chapter 5, *Flash Remoting and ColdFusion MX*

ColdFusion is perhaps the easiest and most popular of the different server-side models available to Flash Remoting, and this chapter shows various techniques for using Flash Remoting with ColdFusion.

Chapter 6, *Server-Side ActionScript*

Server-Side ActionScript (SSAS) allows Flash MX developers to develop server-side code using the familiar ActionScript syntax instead of ColdFusion or Java. ColdFusion and JRun servers can both execute Server-Side ActionScript using a built-in parser based on the open source Rhino implementation of JavaScript.

Chapter 7, *Flash Remoting and Java*

Flash Remoting lets developers enhance any J2EE (i.e., Java) application. This chapter covers applications that run on supported J2EE servers, including Macromedia's JRun 4 server and IBM's WebSphere.

Chapter 8, *Flash Remoting and .NET*

Microsoft's Active Server Pages (ASP) technology is being phased out in favor of ASP.NET. Flash Remoting works well with ASP.NET but does not support ASP. This chapter covers the idiosyncrasies of the ASP.NET implementation by building several real-world examples.

Chapter 9, *Flash Remoting and PHP*

The PHP technology is an open source server platform that attracts many users. Macromedia did not release a version of Flash Remoting for PHP, but a group of developers have created an open source solution. This chapter covers the PHP implementation, AMFPHP.

Part III: Advanced Flash Remoting

Part III covers advanced Flash Remoting techniques.

Chapter 10, *Calling Web Services from Flash Remoting*

Flash Remoting allows Flash to interface with SOAP-based web services, regardless of the server-side language with which the web service is implemented. This chapter covers both basic and advanced techniques for publishing and communicating with remote web services.

Chapter 11, *Extending Objects and UI Controls*

Flash's built-in UI components and ActionScript objects can be extended to offer more functionality. This chapter examines several objects and components and explains how to extend functionality to enhance Flash Remoting.

Chapter 12, *Flash Remoting Best Practices*

Expands on earlier chapters by showing some common best practices that an ActionScript programmer can use with Flash Remoting, such as handling server-side results and errors, organizing code, and clearly separating the UI from the server-side functionality.

Chapter 13, *Testing and Debugging*

Flash supports important debugging tools for serious application development. This chapter explains the ActionScript and NetServices debuggers, as well as general techniques for both client-side and server-side debugging.

Chapter 14, *Real-World Application*

Demonstrates building a simple script repository, including the inserting, updating, deleting, and displaying of data from a database and the uploading, downloading, and storing of scripts.

Chapter 15, *Flash Remoting API*

A Flash Remoting API reference that you will refer to as you build your Flash Remoting applications.

Part IV: Appendixes

Appendix A, *ActionScript Datatype Conversion*

Covers datatype conversion from Flash Remoting to the different server-side languages.

Appendix B, *Books and Online Resources*

Lists other resources covering Flash Remoting or related technologies.

Appendix C, *Specification and Implementation for a Real-World Application*

Includes details of the user interface specification and implementation of the real-world script repository application demonstrated in Chapter 14.

Terminology Conventions

The word *Flash* has different meaning in different contexts, such as a browser plugin or the authoring environment. This book follows these naming conventions:

Flash

Refers generically to Flash movies and the Flash technology.

Flash MX

Refers to the Flash MX authoring environment.

Flash MX 2004 (Flash 2004)

Refers to the Flash 2004 authoring environment. Flash 2004, released in September 2003, is the sequel to Flash MX. Flash Remoting in Flash 2004 is the same as Flash Remoting in Flash MX.

Flash MX Professional 2004 (Flash Pro)

Refers to the Flash Pro authoring environment. Flash Pro offers all the features of Flash 2004, plus a forms-based UI and some additional components. Flash Remoting in Flash Pro is the same as Flash Remoting in Flash 2004 and Flash MX, although there is additional support for access to SOAP-based web services.

Flash Player 7, Flash Player 6, or Flash Player

Refers to the browser plugin that allows users play *.swf* files (movies). I use the term *plugin*, even though the Flash Player is an ActiveX control in Internet Explorer. If we use the generic term *Flash Player* in relation to Flash Remoting, you can assume we mean Flash Player 6 or later. Flash Player 7 has built-in support for SOAP-based web services without requiring a server-side installation.

Flash 5

Refers to the Flash 5 authoring environment. Flash 5 does not support Flash Remoting.

Flash Player 5

Refers to the earlier browser plugin. Flash Player 5 does not support Flash Remoting.

Standalone Player

Refers to the Flash Player that runs on the local system without needing the browser. The Standalone Player for Flash MX or later supports Flash Remoting, but earlier Standalone Players (i.e., Flash 5 and earlier) do not.

Flash Remoting

Refers generically to the technology that includes the ability to communicate between Flash and some server-side component, including the server-side software that supports Flash Remoting and the AMF message format.

Flash Remoting MX and Flash Remoting gateway

Both terms refer to the server-side modules needed for each server model. For ColdFusion, it is a native service (a Java servlet); in ASP.NET it's an assembly; in Java servers, it runs as a servlet; in PHP, it is a class.

Flash Remoting Components

Refers to the downloadable components from Macromedia that you have to install into the Flash authoring environment in order to work with Flash Remoting.

ColdFusion MX

Refers to the Macromedia ColdFusion MX application server.

ColdFusion Markup Language (CFML) and ColdFusion

Both terms refer to the ColdFusion programming language in which *.cfm* pages are implemented.

J2EE server

Refers to Java 2 Enterprise Edition servers, such as JRun 4 or IBM's WebSphere.

Java server

Refers to other implementations of Java servlet technologies, such as Tomcat, that don't necessarily support the J2EE specification.

Java

Refers to the programming language used to build J2EE and Java server applications.

Typographical Conventions

This book follows these typographical conventions:

Menu options

These are shown separated by the → character, such as File → New or Modify → Align → To Stage.

Constant width

Used for directives, variable names, property names, parameter names, ColdFusion markup tags, and code samples. It also denotes code within the text.

Italic

Italics indicate function names, object names, class names, directory names, commands, filenames, and file suffixes such as *.swf* or URLs such as *http://www.macromedia.com/index.html*. Italics are also used for emphasis, such as when introducing a new term to the reader. Functions or methods will usually be followed by parenthesis, like *myFunction()*.

Constant width bold

Indicates that the reader should enter something verbatim. It is sometimes used within code samples for emphasis.

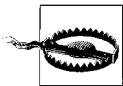
Constant width italic

Indicates a placeholder that should be substituted with your own value, such as *myServiceName*. It can also be used for emphasis within code comments. Optional items are also sometimes shown in square brackets.

Some sections of text that require special attention will be set apart from the text with the following icons:



This is a tip. A tip either reiterates an important point, offers a tidbit of information that relates to the current discussion, or brings to your attention an item that is crucial to understanding the topic at hand.



This is a warning. A warning describes some aspect of the current topic that needs careful attention so that you can avoid possible problems down the road.

We'd Like to Hear from You

The book-writing process is long and arduous, and the examples have been tested and retested. However, mistakes do creep in from time to time. If you find any errors in the text or code examples, please write to:

O'Reilly & Associates, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
1-800-998-9938 (in the United States or Canada)
1-707-829-0515 (international/local)
1-707-829-0104 (fax)

We have a web page for the book, where we list any additional information. You can access this page at:

<http://www.oreilly.com/catalog/flashremoting/>

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, conferences, software, Resource Centers, and the O'Reilly Network, see our web site at:

<http://www.oreilly.com>

In addition to the O'Reilly web site, the author maintains a web site that ties in with the book. There you will find articles, examples, and links to other resources. You can download working versions of most of the examples in the book from the online Code Depot on the author's web site, which is located at:

<http://www.flash-remoting.com>

You can reach the author at:

tom@flash-remoting.com

Future Versions of Flash

The examples in this book were tested most heavily in Flash Player 6 and Flash MX; however, Flash Player 7, Flash 2004, and Flash Pro were still in final beta at the time of this writing. Any discrepancies between Flash MX (and Flash Player 6) and later versions will be noted on the book's accompanying web site: *<http://www.flash-remoting.com>*.

Acknowledgments

Although I've written several books dealing with Macromedia tools, this has been the most difficult, due to the cutting-edge technologies being used. When we first

began, Flash Remoting was available only for ColdFusion, with ASP.NET and Java versions in beta. Now, open source projects are springing up that utilize the AMF format and the Flash Remoting interface for PHP, Java, and Perl.

There are several people who have to be thanked for their generous contributions to the text and code contained in the book. First and foremost, my editor, Bruce Epstein, has gone above and beyond what is the norm for editing a book in this field. He seems to live and breathe for the sake of the books he edits, and has substantially improved the text through constant suggestions, comments, and revisions. Thanks Bruce!

Thanks are due to everyone at O'Reilly who contributed to the publication of this book, including the production editor, Genevieve d'Entremont; Brian Sawyer, who proofread the book; and Julie Hawks, who produced the final index. Likewise, my thanks to Rob Romano for creating the figures from my napkin sketches and for enhancing the screenshots.

The book was started by two of Macromedia's employees: Mike Chambers (Flash Community Manager) and Christian Cantrell (Server Community Manager). Parts of Mike's Chapter 1 still remain, as do parts of Christian's Chapter 5. They provided the seed that started the book.

Several other authors have greatly enhanced the book. Alon J. Salant, a Java expert at Carbon Five (<http://www.carbonfive.com>) who is known for the ASTranslator project, wrote Chapter 7. Joel Martinez wrote Chapter 8 covering ASP.NET based in part on a draft from Jason Michael Perry. Branden Hall, a well-known Flash guru and co-author of *Object-Oriented ActionScript* (New Riders), contributed Chapter 9. Devon H. O'Dell contributed PHP examples in Chapters 1 and 2. Thanks to all five of these gentlemen for making this a well-rounded book.

The book went through several rounds of technical editing to ensure the accuracy of the content. I'd like to thank all of the tech editors who added their comments and improved the book: Marc Garrett, Ray West, Massimo Foti, Sham Bhangal, Joel Martinez, Alon J. Salant, Devon H. O'Dell, Joey Lott, Chafic Kazoun, Jesse Warden, and Robin Debreuil. A special thanks to my usual writing partner Ray West, who was putting together Community MX and organizing several TODCON conferences while I wrote this book. Ray is a good guy and a great partner.

On a personal level, my family has always provided the inspiration for me to keep writing. My wife Janet is the best partner I could have. If this were a novel and I had written a character as a representation of my dream girl, she would be it. Her constant encouragement throughout the process kept me going even when the workload was overwhelming. My daughter Amber has taken a strong liking to Flash and helped with a lot of the visual aspects of Flash that would have completely escaped me had she not been there to help. Thanks Janet and Amber.

—Tom Muck
Fairfax, VA
July 2003

Remoting Fundamentals

Part I covers the setup and installation of Flash Remoting, as well as offering a basic introduction to its use. It also offers details on core concepts and Remoting internals. By the end of Part I, you will have a working knowledge of Flash Remoting, plus a deeper understanding that will act as a springboard for future development.

- Chapter 1, *Introduction to Flash Remoting*
- Chapter 2, *Installing, Configuring, and Using Flash Remoting*
- Chapter 3, *Client/Server Interaction, UI Components, and RecordSets*
- Chapter 4, *Flash Remoting Internals*

Introduction to Flash Remoting

When I was an 11-year-old kid, I thought I was pretty good on a bike. I could do wheelies around the neighborhood, drive on dirt hills, jump ramps. It wasn't enough, though. I wanted to make the transition to a minibike, which is basically a little bike with a lawn mower engine on it. If all I needed was my riding skills, I probably would have been set. Unfortunately, keeping my feet in one place, turning the throttle, and pressing the hand brakes on the minibike were unknown territories. My first time out I turned the throttle too far, lost my footing, and forgot how to hit the brake. I landed in a heap in the street.

If I had put a little forethought into it, I would have realized that riding a bike and riding a minibike were completely different things. Only some aspects were the same; by learning the new aspects and applying my prior knowledge of riding a bike, I eventually figured out how to stay put without falling on my butt.

Working with a new computer technology in a familiar environment is like that. In the case of Flash Remoting, everyone holding this book is probably familiar with Flash movies and ActionScript programming. Flash Remoting puts some new and exciting things into Flash that will require learning new ways to look at ActionScript and what it can accomplish. With Flash Remoting, Macromedia has put an engine on Flash.

What Is Flash Remoting?

These are exciting times. Macromedia is attempting to change the way application developers create web-based applications.

Flash Remoting constitutes a complete rethinking of how web applications are constructed. Using Flash Remoting, you can create complex client/server applications that more closely resemble desktop applications than traditional web pages. For those of you familiar with traditional Windows-style programming IDEs, Flash Remoting is roughly the web equivalent of a Visual Basic client/server application. This chapter presents a broad overview of Flash Remoting.

Flash Remoting technology is at the center of Macromedia's Studio MX product suite, linking the server platforms with the client-side tools. Flash Remoting is built into Macromedia's two application servers—ColdFusion MX and JRun 4. In addition, programmers using other technologies, such as ASP.NET or J2EE application servers, can purchase the Flash Remoting MX package so that Flash Remoting can be utilized on those servers. Although Macromedia doesn't officially support other languages, open source implementations are available for PHP (the AMFPHP project at <http://www.amfphp.org>) and Perl (the FLAP project at <http://www.simonf.com/flap>). There is also an open source Flash Remoting for Java implementation (OpenAMF at <http://www.openamf.org>).

The Flash authoring environment includes UI components, which form the basis of Macromedia's strategy of using a Flash movie as the client in a client/server atmosphere. This approach allows the application server to provide the programming power for the application. Using Flash Remoting, a Flash movie can act as the interface for diverse applications, including connections to databases, SMTP mailers, server components, web services, and much more. Flash Remoting ties together the Macromedia authoring tools and application server technologies to form a new approach to web application development and deployment.

Flash Remoting is a server-side technology that integrates with existing application servers to provide a gateway between the Flash Player and remote services deployed on the server. A *service* can be a simple ColdFusion page or ColdFusion Component (CFC), a PHP or Perl script, a Java class, or an ASP.NET page or DLL. Flash Remoting allows developers to access remote services and web services from within Flash through a simple ActionScript API that is similar to JavaScript. Flash Remoting also allows developers to integrate Flash with existing client/server applications with little modification, to provide a rich, robust user interface that can be deployed across browsers, platforms, and devices.

The Flash Remoting gateway on the server sits between the Flash Player on the client and the server-side tier of an application. It handles data serialization and procedure calls between the Flash Player and the server. *Serialization* is the translation of data to a format that can be easily transmitted over the Web. This translation is transparent to both client-side Flash developers and server-side developers. We'll discuss a client/server architecture using the Flash Player and the Flash Remoting gateway later in this chapter.

Using Flash Remoting, a Flash movie can connect to virtually any remote service deployed on the server. The remote services can be deployed using various languages and technologies, including:

- Java classes
- JavaBeans™
- Enterprise JavaBeans (EJB)

- Java Management Extensions (JMX MBeans)
- ColdFusion templates
- ColdFusion Components
- Server-Side ActionScript (SSAS)
- ASP.NET pages
- ASP.NET DLLs
- SOAP-based web services
- PHP pages

In other words, a remote service might be a ColdFusion page, a PHP page, or an ASP.NET DLL, among other things. Flash Remoting allows Flash to make remote procedure calls on existing server-side services; server-side developers do not have to implement any Flash-specific APIs or adjust their design patterns. Calling a service from a Flash movie is as easy as calling it by name. For example, if you have a server-side method named `getRecords()` that you would typically call from another server-side page to feed an HTML page, Flash Remoting lets you call the `getRecords()` method directly from the Flash movie. Gone are the page reloads associated with HTML pages. The communication with the server is seamless and invisible to the user. This means that the typical web experience is much more user friendly and performs more like a traditional desktop application.

How Does Flash Remoting Work?

The Flash Remoting gateway is installed on the application server and acts as an interface between the Flash Player and the server. The Flash Remoting software that implements the gateway is also called an *adapter*. It has three main tasks:

- Handle requests from the Flash Player to remote services. These services can be on the same server as the Flash Remoting gateway or can be external to the server in the form of web services.
- Translate requests and data from the Flash Player into server-side requests and datatypes.
- Translate responses and data from the server into native ActionScript datatypes.

Figure 1-1 depicts the Flash Player/Flash Remoting architecture.

Communication between the Flash Player and the Flash Remoting gateway is done via HTTP, which has a few implications:

Communication between the Flash Player and the Flash Remoting gateway is request-driven. The Flash Player must initiate all communication with the Flash Remoting gateway. The server cannot push data to Flash unless it is requested by the Flash Player. Use the ActionScript `XMLSocket` object, as discussed in

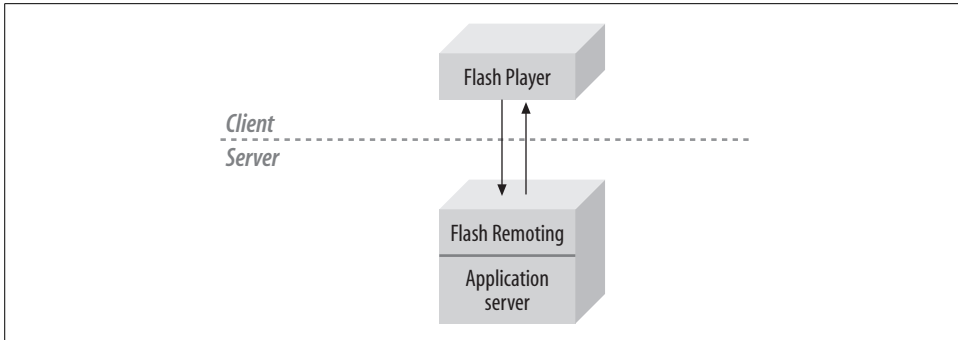


Figure 1-1. The Flash Player/Flash Remoting architecture

ActionScript for Flash MX: The Definitive Guide (O'Reilly), if you need to push data from the server to Flash, such as in a chat application. Another option is to use Macromedia's Flash Communication Server MX (FlashCom) for these types of applications.

HTTP is a stateless protocol, so each request from the Flash Player opens a new connection to the server. The Flash Remoting gateway automatically maintains state between requests through the use of cookies. If cookies are not available on the client, the session state is maintained through a header in the communication packets between the Flash Player and server.

Protocols that work with HTTP, such as SSL, also work with Flash Remoting.

As shown in Figure 1-2, the client/server architecture is the same when the Flash Player communicates with Flash Remoting via HTTPS or SSL as it is using HTTP. Support for HTTPS allows communication between the Flash Player and the server to be encrypted using SSL, provided that the Flash movie is delivered to the client over an SSL connection and displayed within an SSL-enabled browser. This gives a Flash application the same level of security that is available to the HTML application.

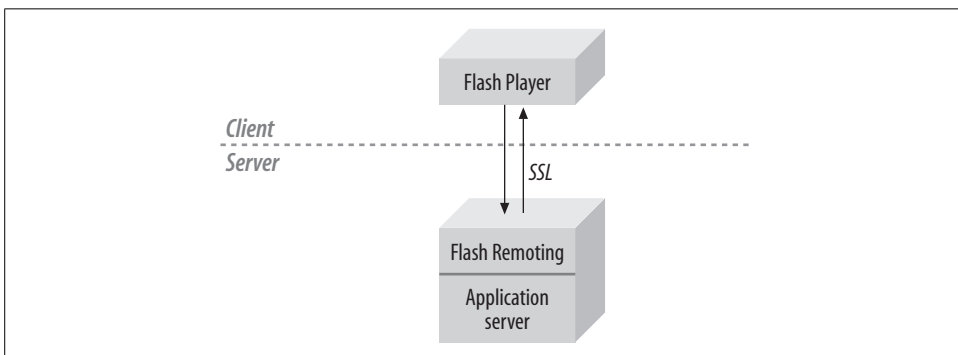


Figure 1-2. Flash Player/Flash Remoting architecture using SSL

Action Message Format

Flash 5 movies could send XML or name/value pairs across HTTP. Although these packets could be parsed automatically by Flash or manually by the developer using custom ActionScript, parsing could be slow because all XML data is sent as text strings encased by cumbersome tags. Flash Remoting is able to handle complex datatypes, such as objects, structures, arrays, and recordsets. A proprietary format was needed to transfer information back and forth between the Flash movie and the application server.

The protocol used for communication between the Flash Remoting gateway and the Flash Player is Action Message Format (AMF). AMF is a binary protocol designed by Macromedia to provide a lightweight, efficient means to serialize, deserialize, and transport data between the Flash Player and the Flash Remoting gateway, as shown in Figure 1-3.

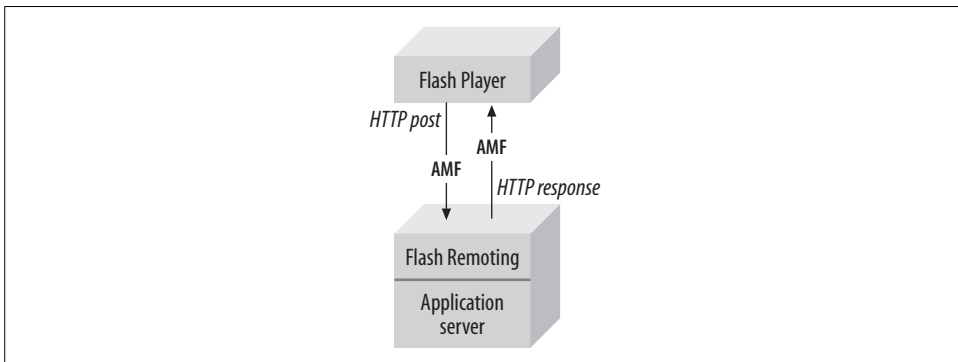
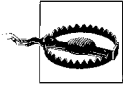


Figure 1-3. Flash Player/AMF/Flash Remoting

It is not necessary to understand AMF in detail to develop robust Flash applications that utilize Flash Remoting; however, it is useful to have a basic understanding of the protocol. Macromedia has not documented the protocol publicly, but the HTTP packets can be examined to gain insight into the format, which seems to be closely based on the format used in remote shared objects (RSOs). The developers of the AMFPHP project have partially documented the format at:

http://amfphp.org/?g=amf_format

The Flash Player communicates with the Flash Remoting gateway via the AMF protocol sent via standard HTTP requests. An AMF packet is sent as a binary POST with the body of the request containing the binary data serialization and remote procedure call information.



Flash Remoting requires browser support for binary POST. Because Netscape 6.x does not support binary POST, Flash Remoting does not work when the Flash Player is running within Netscape 6.x. The Flash Remoting call has no effect and no error is returned. This bug is fixed in Netscape 7. There are also issues with early versions of Safari and Chimera on the Macintosh. You can use a browser-detecting script to redirect users to an appropriate page that uses HTML or older Flash formats if the user's browser does not support your Flash Remoting application.

An AMF packet consists of the following:

- A *packet header*, which describes the AMF version information and includes HTTP headers
- A count of *context headers* in the array that follows
- The *context headers array*, which describes to the server the context in which the information should be processed (on calls from the client) or describes to the client what is coming from the server (a result or status) and its format (such as a recordset or a string)
- A count of the messages in the array that follows
- The *messages array*, which is typically a method call coming from the client or a status/result coming from the server

Following is the HTTP request and response generated by the Hello World examples shown later in this chapter. Because AMF is a binary format, the binary data is shown as periods (...). The ASCII text in the following excerpt, such as `amf_server_debug`, is part of the literal message:

```
POST /flashservices/gateway HTTP/1.1
Referer: http://192.168.0.5:8500/flashservices/gateway
Content-Type: application/x-amf
User-Agent: Shockwave Flash
Host: 127.0.0.1
Content-Length: 198
Cache-Control: no-cache

.....amf_server_debug....pNetDebugConfig
coldfusion...
amfheaders....amf....
httpheaders....recordset....error....trace....m_debug.....
....*com.oreilly.helloworld.HelloWorld.sayHello.../1....
```

And the response from the server back to the Flash Player is as follows:

```
HTTP/1.0 200 OK
Date: Thu, 23 May 2002 02:53:09 GMT
Content-Type: application/x-amf
Content-Length: 69
Server: JRun Web Server
```

```
.....  
/1/onResult...null.....  
Hello World from ColdFusion Component
```

We can gain some insight into AMF by examining the request and response:

- The content type of AMF is `application/x-amf`.
- The AMF data is contained within the body of the request and response; it is primarily binary and is therefore difficult to show in print.
- There are human-readable strings within the AMF data.

The last point tells us that AMF is not entirely compressed. However, even uncompressed, the protocol is very efficient at serializing ActionScript data.

AMF has a number of advantages over traditional Flash data serialization techniques, such as XML and URL-encoded query strings, including:

- AMF is a binary format and thus creates serialized data that is smaller than using string-based encoding. This translates into lower bandwidth requirements and faster loading and response times.
- AMF was built specifically with Flash ActionScript datatypes in mind. Therefore, it can be serialized quickly and efficiently from ActionScript objects and deserialized into ActionScript objects within the Flash Player. In almost all cases, this leads to significant performance increases over string-based serialization.

Why does Flash Remoting use AMF instead of SOAP to communicate with the Flash Player? After all, SOAP was designed as a lightweight protocol for the exchange of information in a distributed environment, which sounds similar to the goals for AMF. Both SOAP and AMF can transfer data and make calls on remote services, and both work over standard HTTP and HTTPS. There are several reasons why Macromedia developed AMF instead of using SOAP:

- SOAP is implemented as XML and is therefore rather verbose compared to the binary AMF.
- AMF is designed and optimized to work with Flash ActionScript datatypes. Deserializing AMF in the Flash Player is much more efficient than parsing and deserializing SOAP, because AMF has direct support for ActionScript datatypes whereas SOAP is a general-purpose protocol. Even if SOAP messages were compressed, serialization in AMF would still be more efficient.
- Adding AMF support to Flash Player 6 required only a small increase in the Player size (about 4 KB compressed), maintaining its slim footprint.
- Integrating full SOAP support on the client side, with acceptable performance, requires an increase in Player size. Although Flash Player 6 includes XML support, it does not support some headers required by SOAP. Using Flash Remoting, Flash can access SOAP-based web services even though Flash Player 6 doesn't support SOAP directly. That is, the Flash Remoting gateway translates SOAP requests to and from AMF format on the server-side and then uses AMF

to communicate with the Flash Player. Flash Player 7 supports SOAP directly, but it will be late 2004 until Flash Player 7 is widely distributed.

The body of the AMF packet contains either an *onResult* event (i.e., *response* event) or an *onStatus* event (i.e., *error* event), which are both ActionScript objects. The Flash movie can then use the object directly, without any further parsing. Chapter 3 contains an in-depth discussion of the *onStatus* and *onResult* events.

Benefits

One of the key benefits of using Flash Remoting over XML or traditional HTML applications is that the application server no longer needs to handle any of the parsing or presentation of information. This frees resources on the server so that it can be better equipped to deal with more complex application logic and/or more users. In addition, session management can be handled on the client inside of the Flash movie rather than on the server. The server still keeps track of the session, but the developer doesn't have to jump through hoops to keep track of users who don't have cookies or track a user session across multiple pages. This equates to huge savings in development time and server resources.

Why Not XML?

I've talked about the benefits of Flash Remoting, but why not use XML? After all, using XML you could encapsulate all of the client/server communication within an ActionScript object and provide a simple API to transfer complex datatypes serialized with XML between Flash and the server. This would have the advantage of not requiring a server-side gateway and would work with Flash Player 5.

The main advantage of Flash Remoting over XML is that it relieves the developer from writing an entire layer of code on both the client and server. XML parsing is built into many of the popular server technologies, but it is cumbersome at best. However, Flash Remoting also has a number of additional advantages:

- It automatically handles all datatype conversions between ActionScript and the server.
- It can convert multiple complex datatypes.
- It seamlessly supports multiple server-side technologies and application servers.
- It allows remote services and web services to be called directly from Flash without requiring any additional server-side code to be written.
- It provides a simple and consistent API for calling remote services and web services from the Flash Player.
- It uses AMF to serialize data, which offers better performance than string-based serialization techniques (such as XML), even though AMF is not as widely supported as XML.

In a typical scenario involving an XML object being sent from a Flash 5 movie to a ColdFusion page, the Flash movie first has to create the XML string manually. Then it has to send the XML string to the ColdFusion page, which has to parse the XML before being able to utilize it. In addition, the server has to transform the result of any operation on the server back into XML to send the result to the Flash movie. The Flash movie then has to parse this XML once again to use the returned information. All of this parsing of data eats up valuable resources and bandwidth even before the application logic can be utilized.

In other words, a Flash 5 movie can't use the data directly from the application server, and the application server can't use the data directly from Flash 5.

Take a typical example of a username and password login. The ActionScript for the Flash movie could create a simple XML string and pass it to the ColdFusion page:

```
// Set up a new XML object.
var returnXML = new XML();
returnXML.ignoreWhite = true;

// Set the callback function for the response from the server.
returnXML.onLoad = handleReply;

// Create an XML string.
// Form field variables are replaced in the string.
var my_xml = '<?xml version="1.0" encoding="iso-8859-1"?>';
my_xml += '<myValidation>';
my_xml += '<username>' + username + '</username>';
my_xml += '<password>' + password + '</password>';
my_xml += '</myValidation>';
var flash_xml_object = new XML(my_xml)

// Send it to the server and then load it into the my_xml object.
flash_xml_object.sendAndLoad("http://192.168.0.4/myLogin.cfm", returnXML);

function handleReply (result) {
    if (result) {
        if (this.firstChild.attributes.logged == "1") {
            greeting = "Hello " + this.firstChild.attributes.username;
            greeting += ". Login was successful";
        } else {
            greeting = "Login failed";
        }
    } else {
        greeting = "There was a communication failure.";
    }
}
```

A ColdFusion page to handle the logic would look like this:

```
<!--Deserialize the username and password from the XML-->
<cfset logged="1">
<cfset my_xml = XMLParse(URLEncode(GetHttpRequestData().content))>
<cfset username = my_xml.myValidation.username.xmltext>
<cfset password = my_xml.myValidation.password.xmltext>
```

```

<!--Query the database for matching entries-->
<cfquery name="myLogin" datasource="myDatasource">
    SELECT username FROM Users
    WHERE username = '#username#' and password = '#password#'
</cfquery>

<!--Check whether a match was found-->
<cfif myLogin.RecordCount LT 1>
    <cfset logged = "0">
</cfif>

<!--Create an XML string to return to the Flash movie-->
<cfset returnXML = "<return username=''" & username>
<cfset returnXML = returnXML & "'' logged=''" & logged & "'' />">
<cfoutput>#returnXML#</cfoutput>

```

As you can see, the code is not intuitive—the XML is manually serialized into a string on both client and server, and the string has to be deserialized and turned into an XML object again on the Flash side. All of this code was created to send one simple XML object from the Flash movie to the server and back again. Imagine if this were something more complex, such as a recordset with 10 or 15 fields and 1,000 rows.

The Flash Remoting version of the previous code might look like this:

```

var myURL = "http://127.0.0.1/flashservices/gateway";
var myServer = NetServices.createGatewayConnection(myURL);
var myService = myServer.getService("com.oreilly.frdg.authentication", this);

myService.getLogin(username, password);

// The result handler for the getLogin() method invocation
function getLogin_Result(result_rs) {
    if (result_rs.getLength() < 1) {
        greeting = "Login failed";
    } else {
        greeting = "Hello " + result_rs.getItemAt(0).username;
        greeting += ". Login was successful";
    }
}

```

And the server-side code might look like this:

```

<cfcomponent displayName="login">
    <cffunction name="getLogin" returnType="query" access="remote">
        <cfargument name="username" type="string">
        <cfargument name="password" type="string">
        <cfquery name="myLogin" datasource="myDatasource">
            SELECT username FROM Users
            WHERE username = '#username#' and password = '#password#'
        </cfquery>
        <cfreturn myLogin>
    </cffunction>
</cfcomponent>

```


The Flash Remoting code in this version is more intuitive; it defines a component containing a function that accepts two arguments directly from Flash. There is no manual parsing; the arguments to the function are passed as strings and a recordset is returned. If the recordset contained 15 fields and 1,000 rows, the server-side code would not look much different.

Using Flash Remoting is much simpler because the Flash movie does not have to package the request in any special format such as XML. Likewise, the ColdFusion Server does not have to package the result for the Flash movie. The data is simply passed back and forth as is and put to use. It is the difference between having a pizza delivered and making the dough and baking the pizza yourself. If the pizza is delivered, the only action you have to take on the pizza is to eat it.

Manually serializing and deserializing data has the advantage of working with Flash Player 5. However, when you consider the amount of client- and server-side code that you have to write, debug, and maintain just to provide basic support for serializing one datatype, the advantages of Flash Remoting become clearer. Considering all the datatypes that Flash Remoting supports and the fact that you can call remote services by name without writing any extra server-side code, Flash Remoting becomes quite attractive.

HTML and Server-Side Code

A typical HTML/server-side template application has problems similar to those in our XML example. The server-side code does not simply perform logic and return information; in many cases, it formats the data as well. Take this simple ColdFusion snippet as an example:

```
<!--Query the database for matching entries-->
<cfquery name="rsGetSearchResults" datasource="bookstore">
SELECT Title, Category, Pub_No FROM Books
WHERE Title LIKE '%#{form.searchfield}%'
</cfquery>
<!--Create an HTML table to display the matches-->
<table border="1">
  <tr>
    <td>Title</td>
    <td>Category</td>
    <td>Pub_No</td>
  </tr>
  <cfoutput query="rsGetSearchResults"
    startRow="#StartRow_rsGetSearchResults#"
    maxRows="#MaxRows_rsGetSearchResults#">
    <tr>
      <td>#rsGetSearchResults.Title#</td>
      <td>#rsGetSearchResults.Category#</td>
      <td>#rsGetSearchResults.Pub_No#</td>
    </tr>
  </cfoutput>
</table>
```

This example queries a database and outputs the results as a table in the browser. This type of mixture of HTML and server-side code is commonplace in web application development. Notice, however, that the presentation of the content is created entirely by the application server. The HTML table doesn't exist until the query is executed and the results are sent to the browser.

Using Flash Remoting, the application server code is utilized for the logic only—querying the database. The results of the query are returned to the Flash movie without any further parsing or manipulating. Using Flash Remoting, a ColdFusion component could be written as follows:

```
<cfcomponent displayName="searchBooks">
  <cffunction name="getTitles" returnType="query" access="remote">
    <cfargument name="search" type="string" default="" />
    <cfquery name="rsGetSearchResults" datasource="bookstore">
      SELECT Title, Category, Pub_No FROM Books
      WHERE Title LIKE '%#form.searchfield#%'
    </cfquery>
    <cfreturn rsGetSearchResults />
  </cffunction>
</cfcomponent>
```

Notice the line in bold, `<cfreturn rsGetSearchResults />`, which returns the entire recordset to the Flash movie as an ActionScript *RecordSet* object. The Flash movie, in turn, can use the recordset without any further parsing. For example, to attach the recordset to a DataGrid component in Flash, you can simply use this result handler in ActionScript:

```
function rsGetSearchResults_Result (result_rs) {
  myGridComponent.setDataProvider(result_rs);
}
```

Again, the Flash movie is working with the recordset as it comes from the server. No further parsing is necessary. Also, the ActionScript programmer has at his disposal a series of highly complex and interactive interface elements, unlike HTML forms, which are limited in functionality.

Another advantage is that the Flash movie looks the same in all browsers. The HTML language is ubiquitous, but the implementation is not uniform. Typically, you have to rely on CSS implementation, JavaScript being enabled, and/or cookies being enabled, or you have to create even more client-side code to handle the many possible user configurations.

Session Management in Flash

HTTP is a stateless protocol. The web server treats each page request coming from a browser as coming from an entirely new user. To create the illusion of maintaining state, many application servers have *state management* (or *session management*) in place to create a seamless experience for the end user. Session management is a bit of

an art form for the application developer, as there are no sure-fire, out-of-the-box methods to maintain state in most application servers.

Typically, in a web application, you use cookies in conjunction with session variables to maintain state. This method won't work if cookies are turned off on the client's browser; therefore, you must store the information in a database or text file for 100% reliability. Furthermore, session management and variables eat up server memory. In an ASP application, for example, each typical user session consumes at least 4.5 KB, unless sessions are explicitly turned off. Also, unless a server application is cluster-aware, managing session state across a cluster can become complicated (because different servers in the cluster might handle successive requests).

Flash Remoting handles session state through means that are invisible to both the user and the server. Session information is sent with each and every AMF packet between the client and the server. No manual session management is required. In a rich client implementation, the Flash movie is loaded into the user's browser only once, so session state is maintained automatically with every call to the server. In addition, because the session state is maintained within the Flash movie in the user's browser, it makes little difference if the application server is clustered.

Flash Remoting Requirements

To develop Flash applications that use Flash Remoting, you must have:

- Macromedia Flash MX or later
- The Macromedia Flash Remoting components
- A server that has the Flash Remoting gateway or the equivalent (such as AMFPHP, FLAP, or OpenAMF)

Macromedia Flash Authoring Tool

The Flash authoring environment is used to create Flash files (.fla and .swf files) and applications. Flash MX was a substantial upgrade to previous versions of Flash. In addition to cosmetic interface changes, the way in which Flash applications are developed has changed. Flash MX or later is required to develop Flash Remoting applications. A fully functional trial version is available from Macromedia at:

<http://www.macromedia.com/software/flash/download/>

You can find more information on Flash at:

<http://www.macromedia.com/software/flash/>



As of Flash MX, Macromedia abandoned a consistent version-numbering scheme for the authoring tool, but the Flash Player is still assigned a numeric version. Flash Remoting requires that you publish your .swf files in Flash 6 format or later.

Director MX, a separate multimedia authoring tool sold by Macromedia, can access Flash Remoting using the Flash MX Asset Xtra (which is basically an embedded version of Flash Player 6). For simplicity, this book assumes you are running Flash in a browser or a standalone Projector.

Macromedia Flash Remoting Components

Flash MX does not come with the Flash Remoting components preinstalled. The Flash Remoting components add support for Flash Remoting to the Flash authoring environment and are required to create Flash files that take advantage of Flash Remoting. If you're using Director MX as the front end, you'll need the Flash MX Asset Xtra, which is compatible with Director MX or Shockwave 8.5.1 or later. The Flash MX Asset Xtra includes Flash Remoting support.

The Flash Remoting components are included with the Flash Remoting gateway and can be found on the Studio MX CD-ROM or downloaded for free from the Macromedia web site. If you're using versions from the Studio MX CD-ROM, make sure you get the latest updates from the Macromedia site as well. The components and other information about Flash Remoting can be found at:

<http://www.macromedia.com/go/flashremoting>

The components make the following items available from within the Flash MX authoring environment:

- ActionScript code and classes necessary for Flash Remoting, including *NetServices.as*, *RecordSet.as*, *DataGlue.as*, and *NetDebugger.as*
- The Remote Service Browser, for examining remote services
- The NetConnection Debugger panel, for debugging Flash Remoting applications
- Flash MX ActionScript editor enhancements for Flash Remoting, and Reference panel documentation

All of these items are discussed in detail in this book, beginning in Chapter 2.

Macromedia Flash Remoting Gateway

The Flash Remoting gateway must be installed on the remote server to allow it to communicate with the Flash movie via Flash Remoting. If you are using an application server that comes with Flash Remoting preinstalled, such as ColdFusion MX or JRun 4, then this step is already taken care of.



Earlier versions of ColdFusion and JRun do not support Flash Remoting.

The Flash Remoting gateway is also sold separately for other application servers, including J2EE servers and Microsoft .NET, and must be installed and configured before it can be used. The trial version of the Flash Remoting gateway for J2EE and ASP.NET can also be downloaded from the Macromedia web site cited earlier.

There are several open source implementations of the Flash Remoting gateway, apart from the versions supported by Macromedia.

The AMFPHP project, discussed in Chapter 9, implements Flash Remoting for PHP application servers:

<http://www.amfphp.org>

The FLAP project implements Flash Remoting for Perl:

<http://www.simonf.com/flap>

The OpenAMF project is an open source alternative to Macromedia's Flash Remoting gateway for Java:

<http://www.openamf.org>

Supported Platforms

This section is a quick summary of Flash Remoting capabilities and server-side services that can be exposed to Flash from supported application servers. For the latest list of supported application servers, platforms, and configurations, see:

<http://www.macromedia.com/go/flashremoting>

Macromedia ColdFusion MX

The Flash Remoting gateway comes preinstalled with Macromedia ColdFusion MX and allows developers to deploy remote services as:

- ColdFusion pages
- ColdFusion Components
- Server-Side ActionScript (SSAS)
- SOAP-based web services

Flash Remoting is also included in the Macromedia ColdFusion MX for J2EE version for deployment on Java application servers, such as WebSphere and BEA WebLogic. It includes support for the remote services supported in ColdFusion MX in addition to those supported on a Java server that ColdFusion is installed on, as discussed below. For more information on ColdFusion MX, or to download a fully functional trial version, go to:

<http://www.macromedia.com/software/coldfusion/>

See also *Programming ColdFusion MX*, by Rob Brooks-Bilson (O'Reilly), for information on ColdFusion. The trial/developer version of ColdFusion MX can be obtained together with Flash MX in the Macromedia Studio MX bundle.

J2EE Application Servers and Java Servlet Engines

The Flash Remoting gateway is available as a standalone product for any J2EE-compatible application server. It also works with a Java servlet engine that has been certified compatible with Sun's servlet 2.2 or 2.3 specifications, such as Tomcat. Flash Remoting for J2EE allows remote services to be deployed as:

- JavaBeans
- Java classes
- Enterprise JavaBeans

Macromedia has tested the functionality with J2EE servers such as JRun 4, IBM WebSphere AS 4, and Sun ONE Web Server, although they also mention reports of success with Tomcat and other servers. For the latest information on supported application servers, consult the Flash Remoting documentation at the Macromedia site.

Chapter 7 discusses Flash Remoting for J2EE servers in detail. The OpenAMF project, an alternative to Macromedia Flash Remoting gateway for J2EE, is also discussed briefly at the end of Chapter 7.

Macromedia JRun 4

JRun is Macromedia's J2EE-compliant Java application server. Flash Remoting comes preinstalled with Macromedia JRun 4 and allows developers to deploy remote services as:

- JavaBeans (stateful)
- Java classes (no state and no pool, new instance on every request)
- Enterprise JavaBeans (EJBHome and EJBObject)
- JMX MBeans
- Server-Side ActionScript (SSAS)
- SOAP-based web services

Because JRun 4 is a J2EE-based server, it supports access to the same services as other J2EE-compatible application servers mentioned earlier, as well as Server-Side ActionScript, JMX MBeans, and SOAP-based web services. For more information on JRun, or to download a fully functional trial version that will revert to a developer's version after 30 days, go to:

<http://www.macromedia.com/software/jrun/>

Microsoft ASP.NET Servers

Flash Remoting is available as a standalone product for Microsoft ASP.NET servers. It allows remote services to be deployed as:

- ASP.NET pages (*.aspx* pages)
- DLL libraries (in the local assembly cache)
- .NET executables
- SOAP-based web services



Flash Remoting does not work with “classic” ASP pages. You must have the ASP.NET framework running on your server. To run the ASP.NET framework you need IIS 5.0 or later, running on Windows 2000 Professional, Windows 2000 Server, Windows XP Professional, or Windows Server 2003 (a.k.a. Windows .NET Server).

Hello World

An introduction to any technology would not be complete without a “Hello World” example. This will give you some hands-on experience with the client-side and server-side code before diving into details. It also provides a sound basis for exploring Flash Remoting on your own.

First, we will look at the Flash code necessary to call the remote service, which is virtually the same regardless of which server-side technology implements the service. We will then look at the server-side code implemented in ColdFusion, Server-Side ActionScript, Java, ASP.NET, PHP, and as a SOAP-based web service.



The examples throughout the book assume that you have Flash Remoting installed and configured on your server, and that you have installed the Flash Remoting components for Flash MX. Chapter 2 covers Flash Remoting installation and configuration in more detail.

Flash ActionScript Code

The client-side ActionScript is virtually the same for each server-side service example. The only things that change are the path to the remote service when it is implemented as a web service and the path to the Flash Remoting gateway, which varies depending on the server implementation.

The client-side ActionScript code shown in Example 1-1 should be inserted on the first frame of the main timeline of a Flash movie, as shown in Figure 1-4.

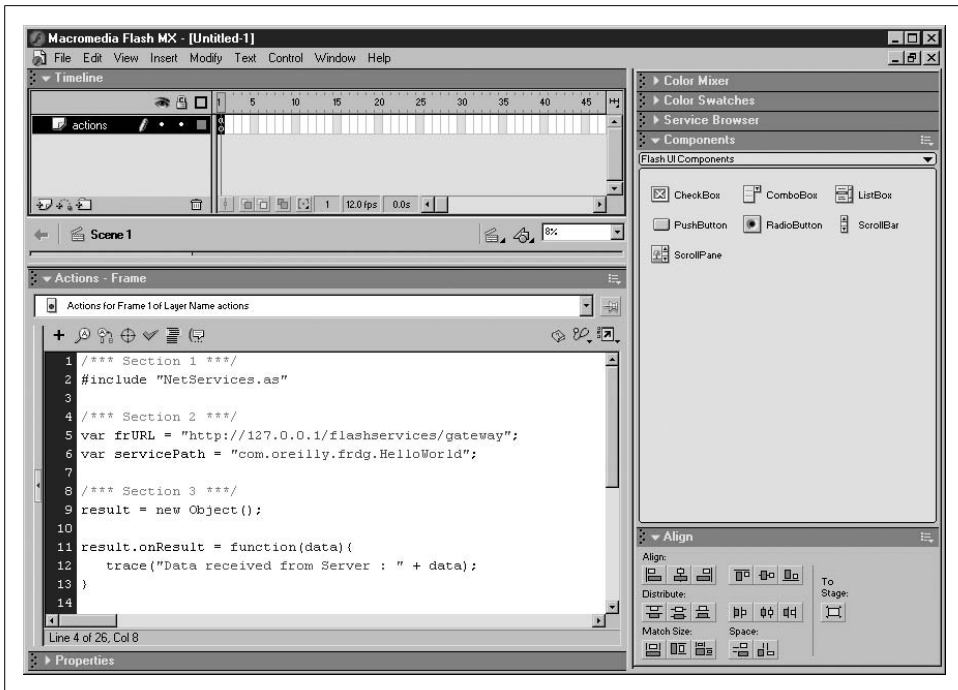


Figure 1-4. Flash timeline with attached client-side ActionScript

Example 1-1. Client-side ActionScript code (HelloWorld.fla)

```

/** Section 1 ***/
#include "NetServices.as"

/** Section 2 ***/
// Assign myURL so it points to your Flash Remoting installation.
var myURL = "http://localhost/flashservices/gateway";
var myServicePath = "com.oreilly.frdg.HelloWorld";

/** Section 3 ***/
myResult = new Object();

myResult.onResult = function (data) {
    trace("Data received from Server : " + data);
};

myResult.onStatus = function (info) {
    trace("An error occurred : " + info.description);
};

System.onStatus = myResult.onStatus;

/** Section 4 ***/
var myServer = NetServices.createGatewayConnection(myURL);

```


Example 1-1. Client-side ActionScript code (HelloWorld.fla) (continued)

```
var myService = myServer.getService(myServicePath, myResult);  
  
myService.sayHello();
```

Section 1 of Example 1-1 includes the *NetServices.as* library, which contains the code necessary to connect to a Flash Remoting-enabled server from Flash. If you do not include *NetServices.as*, the example will not work, but you will not receive any errors within the authoring environment.

Section 2 initializes two variables: *myURL* and *myServicePath*. The *myURL* variable will be used to create a *NetConnection* object that points to the server. The *myServicePath* variable will be used to create a *service object* that points to the service that will be called.

The *myURL* variable specifies the URL to the Flash Remoting gateway installed on the server. If the Flash Remoting gateway is installed on a Microsoft .NET server, the URL will point to the *.aspx* file for the gateway. Similarly, if you are using AMFPHP, the URL will point to a *gateway.php* file on your server.

The *myServicePath* variable specifies the path on the server to the remote service that will be called. The naming convention is similar to a Java package, with each section representing a directory on the server and the last section pointing to the actual service. If the remote service is a Microsoft .NET DLL, *myServicePath* should refer to the DLL's namespace and class name. Similarly, if the remote service is a Java class, the *myServicePath* variable will refer to the package name and class name of the Java class. If the remote service is a web service, *myServicePath* should contain the path to the web service's WSDL file.

Calls from the Flash Player to the application server via the Flash Remoting gateway are asynchronous. Code execution within the Flash Player continues while data is being loaded, which is similar to loading XML into the Flash Player. You must define callback functions, which will be called automatically when the data loads from the server.

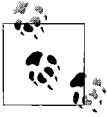


A *callback function* is a function that is called when a specific event occurs. For example, attaching a callback function to an object's *onClick* property causes the callback function to execute whenever the object is clicked. Similarly, a remote service call causes a specific event to occur, which can have a callback function associated with it.

In ActionScript, callback functions can be attached as properties to a generic object (instantiated from the *Object* class). The functions are used to catch data and messages sent back from the server.

Section 3 of Example 1-1 creates an object and attaches two callback functions to it. The *onResult()* callback function is called when data is returned from the remote

service, and the *onStatus()* callback function is called if an error occurs. An object used to receive results from a remote service is called a *responder object* (or sometimes called a *response object*).



Another way to trap events is to specify callback functions named the same as the service name with *_Result* and *_Status* appended to it. This technique, along with more information about callback functions and responder objects, is covered in Chapters 3 and 4.

The *System.onStatus* property specifies the function to be called if the Flash Player cannot connect to the server, as these types of errors are not handled by the *onStatus()* callback function for the remote service call. Example 1-1 sets *System.onStatus* to execute our object's *onStatus()* function. Once we have created an object and the callback functions to receive and process the data returned from the server, we are ready to call the remote service.

Section 4 of Example 1-1 makes a connection to the server by passing in *myURL* (initialized earlier) to the *NetServices.createGatewayConnection()* function. The server connection information is stored in the *myServer* variable. The example then gets a reference to the remote service, which we store in the variable *myService*, by calling the *getService()* method on the *myServer* variable initialized in the previous step. In the call to *getService()*, we pass *myServicePath* to access the desired service and pass our *myResult* object to catch the data or status when the operation completes. We can then use *myService* (the reference to the remote service) to call methods on the service, such as the *sayHello()* method.



The path passed to *getService()*, as specified by *myServicePath*, does not include a file extension for the remote service. Therefore, Flash can access a remote service without knowing its implementation details. One of the powerful aspects of Flash Remoting is that it makes almost all server-side services accessible in a uniform manner. However, you cannot automatically detect which remote services are available. That is, you need to know the remote service methods you intend to call. Flash Remoting has no mechanism in place to find unknown remote services on the fly.

Save the Flash movie as *HelloWorld.fla*. Before the movie can be tested, we need to create the server-side code that implements the *sayHello()* function, as described in subsequent sections.

Example 1-1 utilizes the *trace()* command to display the data in the Output window in the Flash authoring environment. Therefore, the output is visible only when the movie is tested in the authoring environment and not when tested in a browser.

Server-Side Code

In the next section, you'll create the remote service required by this simple Flash movie. Once you have created the remote service, you can test the Flash movie using Control → Test Movie. You should get the following output displayed in the Output window:

Data received from Server : Hello World from *servertype*

If you do not get this result:

- Set the Output window to verbose mode (Window → Output → Options → Debug Level → Verbose).
- Make sure that the server where the Flash Remoting gateway is installed is running and accessible.
- Make sure that there are no syntax errors in your client-side ActionScript code or server-side code.

ColdFusion MX

For the ColdFusion MX example, we will implement the remote service as a *ColdFusion Component* (CFC). CFCs are new to ColdFusion MX and provide an object-based approach to ColdFusion development. They are ideally suited to Flash Remoting. CFCs are discussed in depth in Chapter 5.

Create a file named *HelloWorld.cfc* and place it into the following directory, where *webroot* is the root of your web server and *com\oreilly\frdg* matches the service path specified by the initial portion of the *myServicePath* variable in Example 1-1:

webroot\com\oreilly\frdg

Example 1-2 shows the code that must be added to your *HelloWorld.cfc* component:

Example 1-2. ColdFusion code for HelloWorld.cfc

```
<cfcomponent>
  <cffunction name="sayHello" access="remote" returntype="string">
    <cfreturn "Hello World from ColdFusion Component" />
  </cffunction>
</cfcomponent>
```

This is a simple component that contains one function, *sayHello()*, which returns a string. Notice that we set the access to "remote", which is necessary to allow the component to be called remotely, either by Flash or as a web service.

Save the component. If you have access to the ColdFusion administrative interface (which you should if you have a local installation) browse to it through your browser with the following URL:

http://yourservername/com/oreilly/frdg/HelloWorld.cfc/

After entering your ColdFusion administrative password, you should see a description of the component, similar to Figure 1-5.

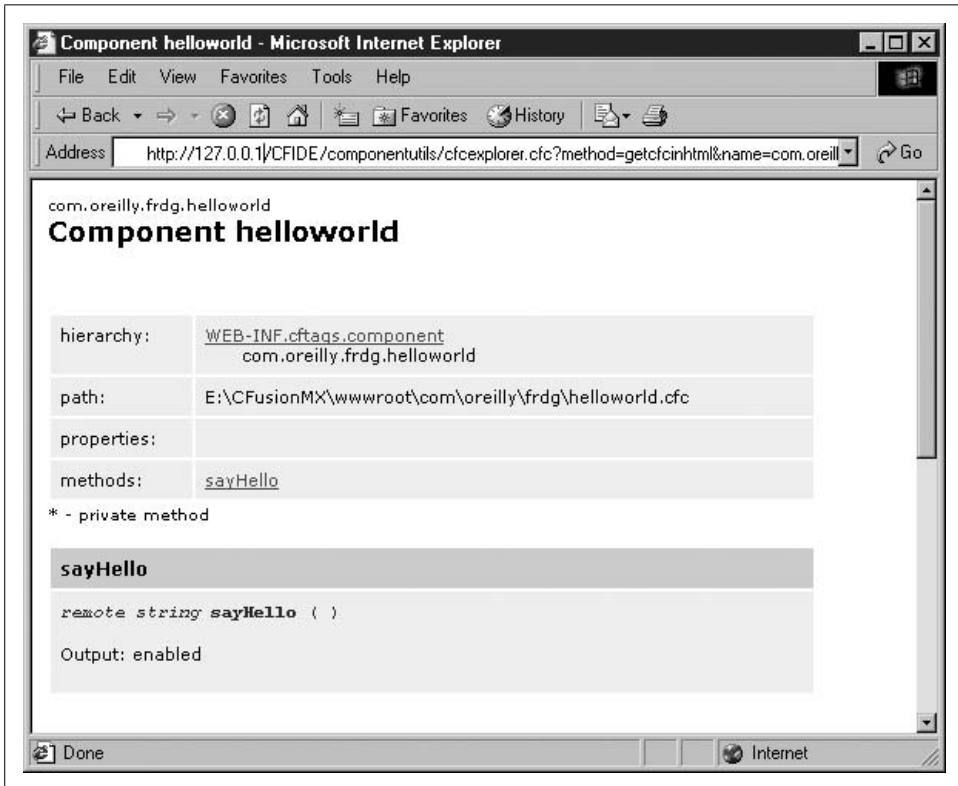


Figure 1-5. ColdFusion MX component description autogenerated by ColdFusion MX

If you do not see the description, or if you get an error, check and fix any syntax errors and try again.

Once you have verified that the ColdFusion component works via the browser, switch back to Flash and test the *HelloWorld.fla* movie created in Example 1-1. You should see “Hello World from ColdFusion Component” in Flash’s Output window.

Server-Side ActionScript

ColdFusion MX and JRun 4 application servers allow developers to create remote services in Server-Side ActionScript (SSAS). *Server-Side ActionScript* is a scripting language that a Flash MX developer can use to create remote services without needing to know a server-side language such as ColdFusion Markup Language (CFML) or Java. Client-side JavaScript and ActionScript programmers may find SSAS easier than learning a new language. Using SSAS, simple services can be written that access databases or utilize the HTTP functionality of ColdFusion or JRun 4. Code written in

SSAS can be consumed by Flash via Flash Remoting only and cannot be used to create other types of output such as HTML.

The SSAS mechanism of ColdFusion MX and JRun 4 is actually a server-side implementation of the Rhino JavaScript parser, with some server-specific objects and methods added that allow the developer access to the functionality of `<cfquery>` and `<cfhttp>` tags of ColdFusion (found in the ActionScript *CF* object). Methods of the *CF* object can be accessed as *CF.methodName()*. You can find a complete discussion of SSAS in Chapter 6. See <http://www.mozilla.org/rhino/> for details on the Rhino project.

To implement the Hello World example in SSAS, create a plain text file named *HelloWorld.asr* using any text editor, and place it into the following directory, where *webroot* is the root of your web server:

webroot/com/oreilly/frdg/



The code in an SSAS (*.asr*) file is not compiled or encrypted. If a user browses to an *.asr* file, the browser displays the code as plain text unless you take steps to prevent it at the web server level. You should turn off read permissions for *.asr* files in your web server or keep the files in a secured directory.

Since ColdFusion can process CFCs, ColdFusion pages, and SSAS files, you need to make sure there are no name conflicts. If you created the ColdFusion component example file earlier, rename *HelloWorld.cfc* to *SomethingElse.cfc* to ensure that the SSAS (*.asr*) file, and not the ColdFusion file, is processed. You may also need to restart the ColdFusion MX server, as the *.cfc* file may have been cached. The exact order in which services are located varies with the application server on which the Flash Remoting gateway is installed. See the appropriate server chapters later in the book for details.

Example 1-3 shows the code that should be added to *HelloWorld.asr*; it creates a simple function called *sayHello()* that returns a string to the client.

Example 1-3. Server-Side ActionScript code for HelloWorld.asr

```
function sayHello () {  
    return "Hello World from Server-Side ActionScript";  
}
```

Save the file in plain text format and switch back to Flash. Test the Flash movie and you should see the output from the SSAS function.

If you get an error saying that the service cannot be found, check the service path, and make sure that there are no syntax errors in the *.asr* file.

Java using JRun 4 or other J2EE servers

For the Java example, we will implement our remote service as a simple Java class. Using Java as a remote service requires that the Flash Remoting gateway be installed on a Java application server such as Macromedia's JRun 4 or IBM's WebSphere. The Java version will not work with ColdFusion MX or Microsoft .NET servers.

Create a new plain text file in any text editor, name it *HelloWorld.java*, and enter the code shown in Example 1-4.

Example 1-4. Java code for HelloWorld.java

```
package com.oreilly.frdg;

public class HelloWorld {
    public String sayHello () {
        return "Hello World from Java";
    }
}
```

Compile the class into your web server's classpath. This may vary from server to server, but the server's *WEB-INF* (or *SERVER-INF* in the case of JRun) directory is usually included within the server's classpath. For example, to compile it using JRun 4, you would use (from a command prompt):

```
c:\jrun4\servers\myservername\server-inf\classes\com\oreilly\frdg>javac
HelloWorld.java
```

If you are using JRun 4 and created the SSAS example earlier, rename *HelloWorld.asr* to *SomethingElse.asr* to ensure that the Java class is used instead.

Once the class has been successfully compiled, place it in the *classpath\com\oreilly\frdg* directory and switch to Flash and test your movie. You should see the output from the *sayHello()* method of the *HelloWorld* Java class. If you get an error that the service cannot be found, make sure that you have compiled the class into the server's classpath.

Microsoft .NET server

ASP.NET services can be written in several languages, including VB.NET and C#. This Microsoft .NET service example is implemented as a .NET DLL written in C#.

Open Microsoft's Visual Studio .NET (VS.NET) and create a new project. From the Project Types window, select Visual C# Projects; then, from the Templates window, select Class Library. Set the name of the project to *HelloWorld*, as shown in Figure 1-6. Rename the class file that appears from *Class1.cs* to *HelloWorld.cs*. The code will work even if you do not rename the class file, but renaming it makes it easier to organize the files.

Example 1-5 shows the server-side C# code to implement the example as a Windows .NET service.

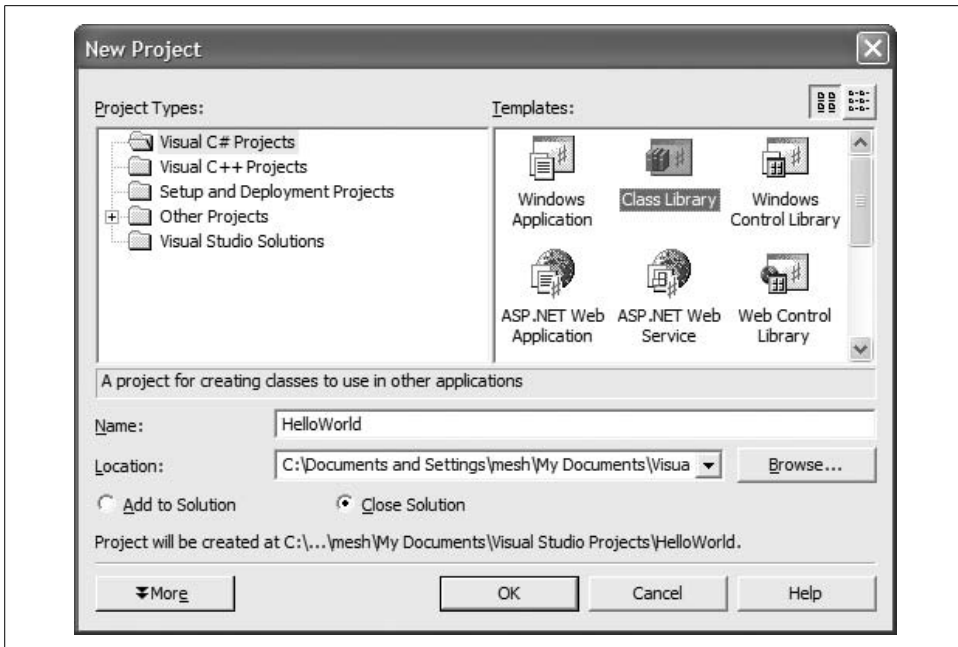


Figure 1-6. Visual Studio .NET project setup screen with settings for HelloWorld DLL

Example 1-5. C# code for HelloWorld.cs

using System;

```
namespace com.oreilly.frdg {
    public class HelloWorld {
        public String sayHello () {
            return "Hello World from ASP.NET DLL";
        }
    }
}
```

Enter the code shown in Example 1-5 and compile the DLL using VS.NET's Build → Build Solution option, which creates *HelloWorld.dll* in the following directory:

projectpath/bin/Debug

Copy *HelloWorld.dll* into the *flashservices/bin* directory on your .NET web server at:

webroot/flashservices/bin/

The DLL contains a class with one function, *sayHello()*, which returns a string. The service path within Flash is determined by the DLL's namespace plus the class containing the method being called. By setting the namespace to the same as the directory structure for our other examples, we will not have to change the *myServicePath* variable within our client-side *ActionScript*. Using a unique namespace also protects your DLL from namespace collisions with other DLLs.

Switch back to the Flash movie and change the `myURL` variable in Example 1-1 to point to the .NET version of the Flash Remoting gateway, such as:

```
var myURL = "http://yourservername/flashremoting/gateway.aspx";
```

This is the only change that has to be made to the Flash movie. It is necessary because the .NET version of the Flash Remoting gateway is implemented differently than the Java and ColdFusion MX versions.

Save the Flash movie and test it. You should see the output from the DLL (“Hello World from ASP.NET DLL”) in Flash’s Output window.

PHP

The Hello World application (and other applications) must be set up a bit differently in PHP than in other environments. Flash Remoting with PHP is class-based, due to requirements of the AMFPHP library. That is to say, all Flash Remoting services must be written as classes in PHP. To install the AMFPHP library, simply download the source release package and copy its *flashservices* directory to your web server’s document root (see Chapter 9 for additional details). Because the class is named *com.oreilly.frdg.HelloWorld*, AMFPHP searches in the services path for a *HelloWorld.php* file. The main *flashservices* directory resides under the web root, with the AMFPHP classes in that directory. The *services* directory resides in this *flashservices* directory as well.

When building PHP remote services, you should include a *gateway.php* file in your server-side application in the directory for your current project. This creates the Flash Remoting gateway and includes the necessary files. The *gateway.php* file (shown in Example 1-6) for the Hello World example should be saved in the *webroot\com\oreilly\frdg* directory.

Example 1-6. PHP Remoting gateway.php file contents

```
<?php
/* File: gateway.php
   Instantiates the Gateway for the HelloWorld Application */
require_once '/app/Gateway.php'; /* Require files */
$gateway = new Gateway();        /* Create the gateway */
$gateway->setBaseClassPath('/services/com/oreilly/frdg');
                                   /* Set the path to where the service lives */
$gateway->service();              /* Start the service */
?>
```

Create a file named *HelloWorld.php* and place it into the following directory, where *webroot* is the root of your web server and *com\oreilly\frdg* matches the service path specified by the initial portion of the `myServicePath` variable in Example 1-1:

webroot\flashservices\services\com\oreilly\frdg

Add the code shown in Example 1-7 to your *HelloWorld.php* page.

Example 1-7. PHP code for HelloWorld.php

```
<?php
/* File: {SERVICES_CLASS_PATH}/com/oreilly/frdg/HelloWorld.php
   provides the HelloWorld class used in Chapter 1. */
class HelloWorld {
    function HelloWorld () {
        $this->methodTable = array(
            'sayHello' => array(
                'description' => 'Says Hello from PHP',
                'access' => 'remote',
                'arguments' => array ('arg1')
            )
        );
    }
    function sayHello () {
        return 'Hello World from PHP';
    }
}
?>
```

Example 1-7 implements a simple class named *HelloWorld* that contains one method, *sayHello()*, which returns a string. The class is named the same as the file. The *methodTable* array is used by AMFPHP to look up functions to invoke and to provide a pseudoimplementation of ColdFusion’s CFCE Explorer utility, which documents the class, methods, properties, arguments, return types, and so forth.

Switch back to the Flash movie and change the *myURL* variable in Example 1-1 to point to the AMFPHP gateway:

```
var myURL = "http://yourservername/com/oreilly/frdg/gateway.php";
```

This is the only change that has to be made to the Flash movie, and it is necessary because the PHP implementation utilizes PHP pages to handle the functionality of the gateway.

If you run the movie in the test environment, you should see the phrase “Hello World from PHP” in the Output window. If you don’t see it, verify that you have correctly installed the AMFPHP classes and verify your code.

Web service

For the web service example, we will create a web service using ColdFusion MX. However, any web service containing a *sayHello()* method that returns a string works just as well.

Creating a web service in ColdFusion MX is extremely simple; we simply pass the URL to our CFC, adding *?wsdl* to the query string, which tells ColdFusion to generate a web service from the component. We’ll use the CFC that we created in Example 1-2, *HelloWorld.cfc*, saved in the directory specified earlier.

Browse to the component with a web browser, and add the *?wsdl* query string to the URL that points to the component:

http://localhost/com/oreilly/frdg/HelloWorld.cfc?wsdl

The browser should display the WSDL XML for the web service, as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://frdg.oreilly.com"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:intf="http://frdg.oreilly.com" xmlns:impl="http://frdg.oreilly.com-impl"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:message name="CFCInvocationException">
    </wsdl:message>
    <wsdl:message name="sayHelloResponse">
      <wsdl:part name="return" type="SOAP-ENC:string"/>
    </wsdl:message>
    <wsdl:message name="sayHelloRequest">
      <wsdl:part name="username" type="SOAP-ENC:string"/>
    </wsdl:message>
    <wsdl:portType name="hellouser">
      <wsdl:operation name="sayHello" parameterOrder="username">
        <wsdl:input message="intf:sayHelloRequest"/>
        <wsdl:output message="intf:sayHelloResponse"/>
        <wsdl:fault name="CFCInvocationException"
          message="intf:CFCInvocationException"/>
      </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="hellouser.cfcSoapBinding" type="intf:hellouser">
      <wsdlsoap:binding style="rpc"
        transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="sayHello">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input>
          <wsdlsoap:body use="encoded"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://frdg.oreilly.com"/>
        </wsdl:input>
        <wsdl:output>
          <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/
            encoding/"
            namespace="http://frdg.oreilly.com"/>
        </wsdl:output>
      </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="hellouserService">
      <wsdl:port name="hellouser.cfc" binding="intf:hellouser.cfcSoapBinding">
        <wsdlsoap:address
          location="http://127.0.0.1/com/oreilly/frdg/hellouser.cfc"/>
        </wsdl:port>
      </wsdl:service>
    </wsdl:definitions>
```

If you see only a blank screen, view the page's source in your browser (using View → Source in Internet Explorer, for example). If you receive an error, correct any errors identified by the error message and try again. Like any URL, the web service URL may be cached depending on the browser settings, so you should reload/refresh the page to make sure the browser isn't using the cached version. This web service can also be seen at the author's site at:

<http://www.flash-remoting.com/oreilly/com/helloworld.cfc?wsdl>

Switch to Flash and change the `myServicePath` variable to point to the web service's WSDL file. If you are using the CFC to create the web service, the path will be:

```
var myServicePath = "http://yourservername/com/oreilly/frdg/HelloWorld.cfc?wsdl";
```

Test your movie, and you should see the output from the `sayHello()` method of the web service. Although our web service is on the same server as the Flash Remoting gateway, Flash Remoting is simply acting as a gateway when accessing an XML-based (SOAP-compliant) web service. The web service can be on any computer accessible via the network or the Internet.

When working with Flash Remoting and web services, you are not limited to ASP.NET, ColdFusion, PHP, and J2EE. Web services can be implemented in:

- Python or Perl
- C or C++
- Any other language that has a SOAP library implementation

More information on web services can be found at:

<http://www.xml.com/webservices>

Overview

The Hello World example, while simple, illustrates the power of using Flash Remoting. The core client-side `ActionScript` code is the same, regardless of the language or server model that the remote service is written in. At most, only the path to the Flash Remoting gateway or remote service is different.

Furthermore, none of the server-side code is Flash-specific. This means that you can create libraries of functions that work from the server-side languages, for use without Flash, which can also be called directly from Flash. In many cases, you will be able to integrate a Flash front end with existing server-side code and libraries with little or no changes on the server. (Details and exceptions are covered throughout the rest of the book.)

Isolation between server-side and client-side code allows for a clean division of labor. Server-side developers need not worry about what is calling their code; if there is a well-defined API on the server, Flash developers can seamlessly hook into the server-side code. Similarly, the Flash developer need not worry about the details of the

server-side implementation. He need only know the API for the remote services he intends to call. If he is using web services, he can query the *.wsdl* file on the server to discover the methods. This allows both the server-side code and the Flash application to be developed simultaneously, reducing production time and making testing and debugging easier.

Even if one developer writes both the Flash and server-side code, the multitiered architecture is still advantageous. It allows you to define an API, implement it on the server, and then hook the Flash movie into it. This makes it possible to test each component on its own before connecting Flash to the server, ensuring that bugs are less frequent and easier to isolate.

Our example may seem simple, because we are only passing a string from the server to Flash. However, if you think of a string as just another type of object or datatype, you can begin to see the power of Flash Remoting. Try passing more complex datatypes, such as an array, from the server-side service to Flash, and see what is returned to the Flash movie. Modify the *onResult()* callback function from Example 1-1 to do something more interesting with the data than display it in the Output window.

Workflow Example

Having discussed how the Flash Player and the Flash Remoting gateway communicate, now let's look at what occurs behind the scenes. We will examine each step of the earlier Hello World example. In Example 1-1, a remote service was called from Flash and received a "Hello World" string in response.

Here are the steps that occur:

1. Using the NetServices API within Flash, developer-written code makes a call for a remote service.
2. The NetServices library passes the remote service call, along with any arguments, to the *NetConnection* object within the Flash Player.
3. The *NetConnection* object serializes the request into AMF and sends it to server as an HTTP binary POST.
4. The Flash Remoting gateway on the server receives the request, deserializes it and determines the server-side service to which to pass the request.
5. The Flash Remoting gateway on the server invokes the server-side service, passing any arguments sent along with the request from the Flash Player.
6. The Flash Remoting gateway on the server receives any data returned from the service (in this case, the string "Hello World"), serializes it into AMF, and returns it to the client-side Flash Player as an HTTP response.

7. The Flash Player receives the AMF data from the server and deserializes it into a native ActionScript datatype (in this case a *String* object). Depending on the data sent back, the deserialization is done within the Flash Player or the NetServices code.
8. Finally, the string is returned to an ActionScript callback function specified by the developer to receive data loaded from the server.

Although a lot happens when a remote service is called from the Flash Player, most steps are abstracted away from the developer. The developer has only to write the client-side ActionScript that calls the remote service (Step 1) and receive the response from the remote service (Step 8). Of course, someone has to write the code for the remote service itself (Step 5), but that is often done by a different developer or independently of Flash, such as in the case where a Flash front end is being added to an existing web service.

Architecture for Flash Remoting Applications

We conclude this chapter with a general overview of a client/server architecture using the Flash Player and the Flash Remoting gateway. Generally, Flash/server applications follow an *n-tiered architecture*. Figure 1-7 depicts such an architecture, comprising a client/presentation tier (the Flash Player), a middle tier (Flash Remoting gateway running within an application server), and a data tier (a database, XML file, or other data source).

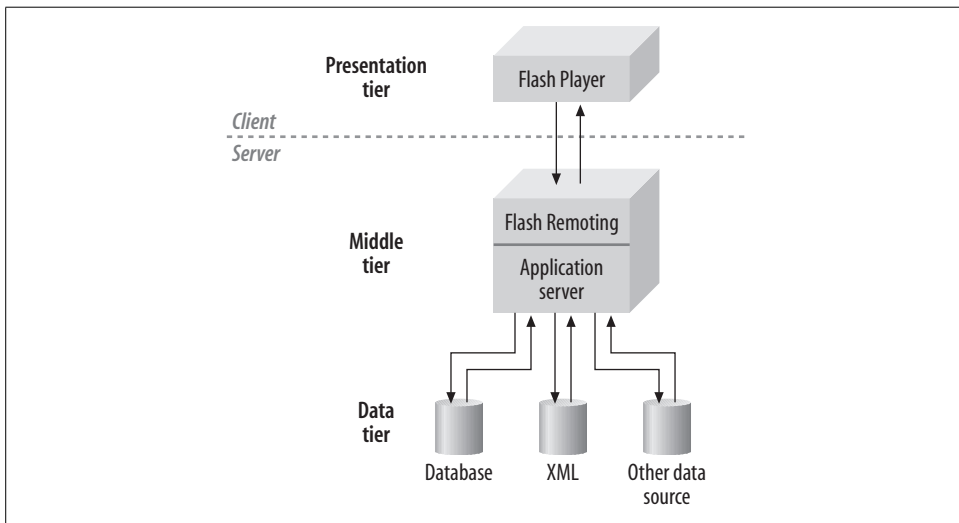


Figure 1-7. The Flash/server *n-tiered application architecture*

Presentation Tier

The presentation tier is responsible for the application's user interface (UI) and any client-side logic that is needed, such as client-side data validation. It communicates with the middle tier by sending and loading data on a request-driven basis. In most cases, the presentation layer consists of a Flash file embedded within an HTML page, but it can also be a Flash Standalone Projector running on the desktop, or even a Flash sprite within a Director Projector.

Middle Tier

The middle tier sits between the presentation layer and the data source. Its primary role is to separate the presentation tier from the data tier and provide access to the data tier from the Flash Player. The core application logic is also normally implemented in the middle tier. This frees the client to do what it does best—presentation—and frees the server to do what it does best—communicate with databases and manipulate data.

The middle tier resides on the server and can be implemented with various technologies, such as ColdFusion, ASP.NET, and Java. In addition, the middle tier can consist of multiple levels, each adding a layer of abstraction from those immediately above and below it.

In Flash 5, it was common to have a multitiered middle layer, with the uppermost layer serializing and deserializing data to and from the Flash Player, as shown in Figure 1-8.

However, using Flash Remoting and Flash Player 6 or later, this logic is handled by the Flash Remoting gateway, sitting on the server atop the middle tier, as shown in Figure 1-9. Flash Remoting eliminates the need to write Flash-specific code in the middle tier.

Data Tier

The data tier is the lowest level of the architecture and is responsible for managing the application's data, as well as the data's persistence. The data source resides on the server side and can be implemented as a database (such as SQL Server, DB2, MySQL, or Oracle), XML file, comma-separated file, and so forth. Client-specific data, such as user interface preferences, can be stored on the client side using cookies or ActionScript local shared objects (LSOs). Whether this data is stored on the client side or in the data tier on the server side depends primarily on the importance of the data to the functionality of the application, with application-critical data generally being stored on the server, which is considered more reliable.

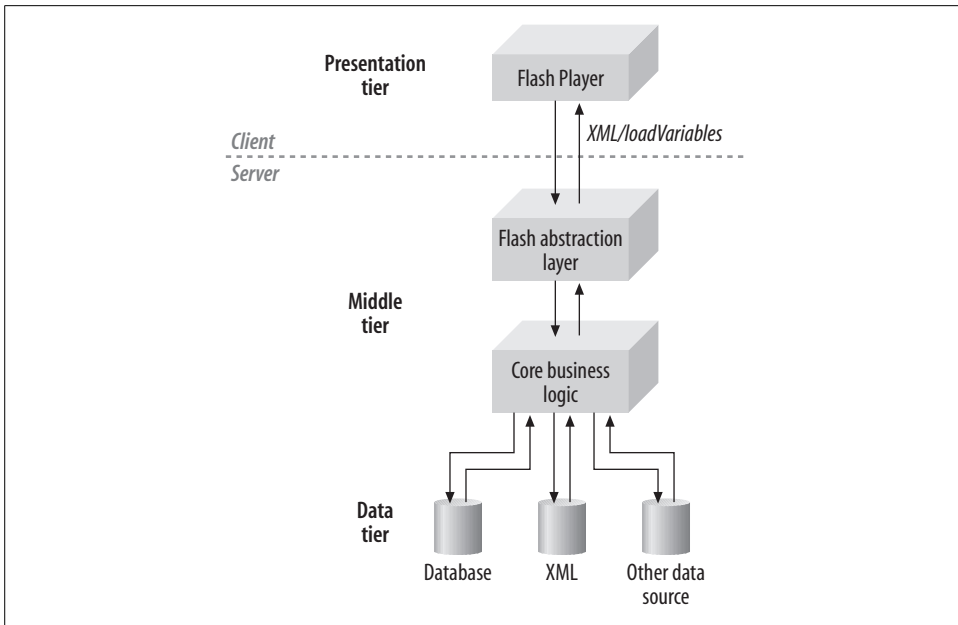


Figure 1-8. Flash 5 n-tiered application architecture with multilayered middle tier

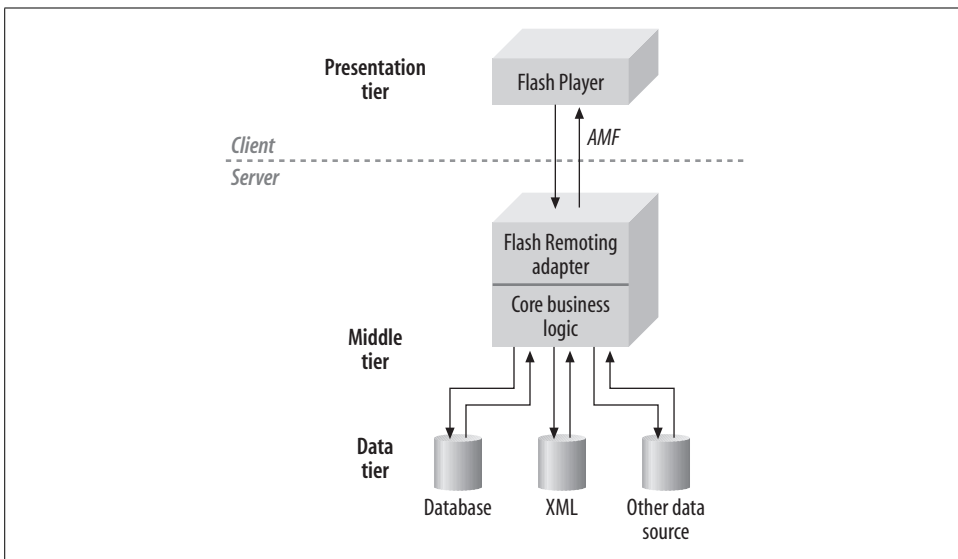


Figure 1-9. N-tiered application architecture with Flash Remoting gateway

This n-tiered architecture is similar to an n-tiered architecture that uses HTML within a web browser as the presentation tier. However, there are some important differences. When using Macromedia Flash, the presentation layer can be generated

entirely on the client side, as the UI can be created dynamically at runtime on the client's machine. Furthermore, once the UI has been created or downloaded, only the updated data has to be sent back and forth to the server. This differs from dynamically generated HTML, which requires that the entire page be recreated on the server and served to the client each time the data or state of the application changes.

There are other advantages of using an n-tiered architecture with the Flash Player as the presentation layer. By abstracting the presentation layer away from the data source, you can completely change the data source or its format without affecting the interface or rewriting your Flash movie.

Changes to the data tier affect only the middle tier. Assuming that the API exposed by the middle tier to the presentation level does not change, the Flash movie will not be affected at all.

For example, the application's data source can be changed without affecting the Flash movie, as shown in Figure 1-10. Furthermore, because the middle tier might comprise multiple levels, only the middle tier's interface to the data tier needs to be adjusted.

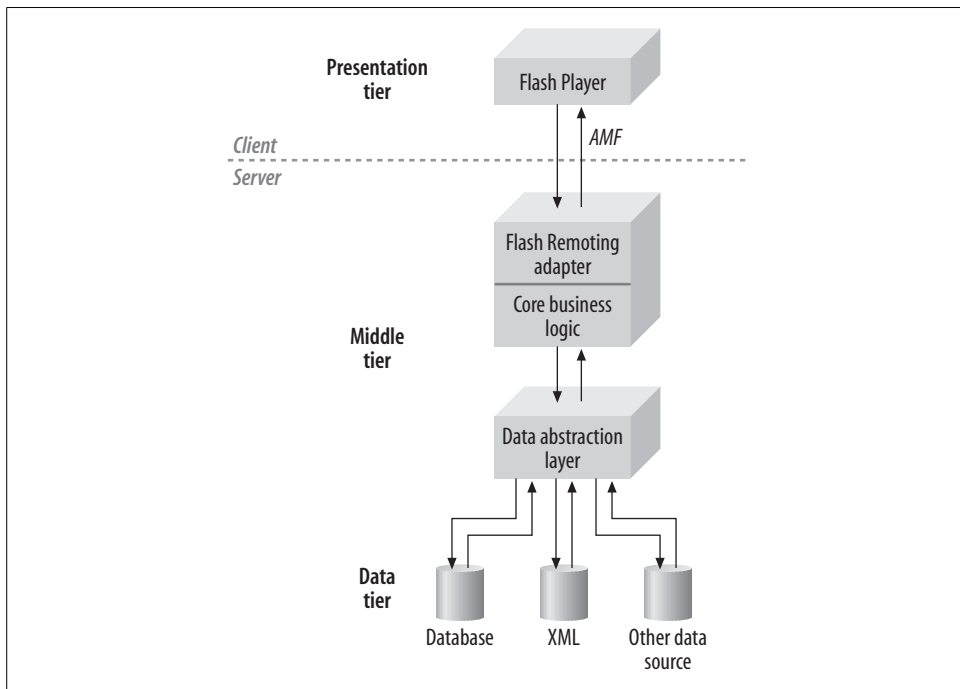


Figure 1-10. N-tiered architecture with data abstraction layer

By having multiple tiers, each tier can be optimized for its particular task and environment. This is particularly important when using Flash as the presentation layer,

since Flash runs on the client side, which can be a much more variable and unknown environment than the server.

For example, you can filter large sets of data in the middle tier where you have a known environment and resources, versus doing it on the client side within Flash where, depending on the client's machine, it might not perform well. In this case, you may want to initially sort the data set on the server and then have any user-initiated sorts occur within the Flash Player. This is a good tradeoff between client-side processing concerns and the extra bandwidth required to transfer data sets to and from the server. Components such as the DataGrid from Macromedia further abstract the implementation of this logic and allow complex sorting and filtering directly within the Flash movie. I address these topics in subsequent chapters, using examples where appropriate.

It is much easier to update the application's core business logic when it is centralized in the middle tier, versus spreading it out across multiple tiers and technologies. Furthermore, keeping business logic out of the presentation tier allows you to test the business logic separate from the presentation tier, isolate any problems, deploy changes, and integrate it with the other tiers of the architecture.

Finally, this multitiered architectural structure mirrors the common divisions of labor during Flash application development. Often, a Flash developer creates the Flash movie and client-side ActionScript, while another developer creates the server-side code. They can program and test their code independently, making development much easier, faster and less error-prone. Provided that the interfaces between the levels are defined, the Flash developer can use temporary data hardcoded into the application to test the application. The server-side developer simply needs to ensure that his code implements the defined API to the presentation tier.



Avoid the temptation to use ActionScript to implement the application's business logic within the Flash movie's presentation layer. Such an approach ties the Flash movie too closely to the lower levels of the architecture and exposes the business logic on the client side, making the application more difficult to maintain and update, as well as possibly affecting client-side performance adversely. When working with Flash Remoting, you are building a *client/server* application, not a standalone Flash movie. The application server is much better suited for the business logic. Chapter 12 goes into much greater detail about the best practices in building a Flash Remoting application.

Again, this is a very broad and general overview of client/server application architecture when using Macromedia Flash for the presentation layer. Later chapters discuss differences specific to additional technologies.