



macromedia®
FLASH™
REMOTINGMX

Using Flash Remoting MX



Trademarks

Afterburner, AppletAce, Attain, Attain Enterprise Learning System, Attain Essentials, Attain Objects for Dreamweaver, Authorware, Authorware Attain, Authorware Interactive Studio, Authorware Star, Authorware Synergy, Backstage, Backstage Designer, Backstage Desktop Studio, Backstage Enterprise Studio, Backstage Internet Studio, ColdFusion, Design in Motion, Director, Director Multimedia Studio, Doc Around the Clock, Dreamweaver, Dreamweaver Attain, Drumbeat, Drumbeat 2000, Extreme 3D, Fireworks, Flash, Fontographer, FreeHand, FreeHand Graphics Studio, Generator, Generator Developer's Studio, Generator Dynamic Graphics Server, JRun, Knowledge Objects, Knowledge Stream, Knowledge Track, Lingo, Live Effects, Macromedia, Macromedia M Logo & Design, Macromedia Flash, Macromedia Xres, Macromind, Macromind Action, MAGIC, Mediamaker, Object Authoring, Power Applets, Priority Access, Roundtrip HTML, Scriptlets, SoundEdit, ShockRave, Shockmachine, Shockwave, Shockwave Remote, Shockwave Internet Studio, Showcase, Tools to Power Your Ideas, Universal Media, Virtuoso, Web Design 101, Whirlwind and Xtra are trademarks of Macromedia, Inc. and may be registered in the United States or in other jurisdictions including internationally. Other product names, logos, designs, titles, words or phrases mentioned within this publication may be trademarks, servicemarks, or tradenames of Macromedia, Inc. or other entities and may be registered in certain jurisdictions including internationally.

This guide contains links to third-party websites that are not under the control of Macromedia, and Macromedia is not responsible for the content on any linked site. If you access a third-party website mentioned in this guide, then you do so at your own risk. Macromedia provides these links only as a convenience, and the inclusion of the link does not imply that Macromedia endorses or accepts any responsibility for the content on those third-party sites.

Copyright © 2002 Macromedia, Inc. All rights reserved. This manual may not be copied, photocopied, reproduced, translated, or converted to any electronic or machine-readable form in whole or in part without prior written approval of Macromedia, Inc. Part Number ZRFCM200

Acknowledgments

Project Management: David Golden

Writing: Stephen M. Gilson, Hal Lichtin, Michael Peterson

Editing: Linda Adler and Noreen Maher

Second Edition: September 2002

Macromedia, Inc.
600 Townsend St.
San Francisco, CA 94103

CONTENTS

ABOUT THIS BOOK	VII
Who should read this book	viii
Developer resources	viii
About Macromedia Flash Remoting MX documentation	ix
Getting answers	ix
Contacting Macromedia	ix
CHAPTER 1 Using Flash Remoting MX	1
About Flash Remoting MX	2
Understanding the Flash Remoting service adapters	3
Understanding AMF	3
Building Flash applications with Flash Remoting MX	5
Understanding the Flash Remoting development environment	5
Applying design patterns to Flash Remoting MX	6
Building a Hello World application with Flash Remoting	9
Building the remote service	9
Calling the remote service from ActionScript	10
CHAPTER 2 Using Flash Remoting Components in ActionScript	13
Flash Remoting application structure	14
Flash Remoting classes	16
About the Flash Remoting classes	17
Including Flash Remoting ActionScript files	18
Configuring Flash Remoting MX	19
Creating the Flash Remoting connection object	19
Creating the service object	22
Authenticating to the application server	23
Calling service functions	25
Specifying a responder object	25
Calling functions using named arguments in ColdFusion	25
Specifying functions	26
Handling service results	27
Result-handling hierarchy	27
Result-handling strategies	27
Handling errors	31
The error object	31

Error-handling hierarchy	31
Error-handling strategies.	32
CHAPTER 3 Using Flash Remoting Data in ActionScript.	35
About Flash Remoting MX and data types.	36
Converting from ActionScript to application server data types.	37
ActionScript data conversion notes.	38
Converting from application server data types to ActionScript.	39
Server data conversion notes.	40
ColdFusion to ActionScript data conversion issues.	41
Working with objects.	43
Working with ActionScript typed objects.	43
Working with serializable Java objects	45
Working with RecordSet objects	46
About record sets	46
RecordSet methods.	47
Using RecordSet methods and properties	48
Delivering RecordSet data to Flash applications in ColdFusion MX.	53
Using Flash MX UI components with RecordSet objects	54
Working with XML.	57
CHAPTER 4 Using the NetConnection Debugger	59
About the NetConnection Debugger.	60
NetConnection events	61
NetConnection event types	61
Common event information.	62
client event messages	63
app_server event messages	64
Flash Communication Server events.	65
Using the NetConnection Debugger in ActionScript.	66
Using the NetDebug.trace method.	66
Using connection-specific debugging methods	66
Configuring debugger output in ActionScript	66
CHAPTER 5 Using Flash Remoting MX with ColdFusion MX . . .	69
Using Flash Remoting MX with ColdFusion pages	70
Determining the Flash service name.	70
Using the Flash scope to pass parameters to ColdFusion pages.	71
Accessing ActionScript objects	73
Using Flash Remoting MX with ColdFusion components.	77
Determining the Flash service name.	77
Returning results to ActionScript	77
Passing parameters to ColdFusion components	78
Accessing ActionScript objects	80
Using component metadata with the Flash Remoting service	81
Using Flash Remoting MX with server-side ActionScript.	82
Using CF.http	82
Using CF.query	85
Calling web services from Flash Remoting MX	86
Invoking web service methods using Flash Remoting MX	86

Securing access to ColdFusion from Flash Remoting MX	88
Handling errors with ColdFusion	90
CHAPTER 6 Using Flash Remoting MX for Java	91
About Flash Remoting MX for Java	92
How does Flash Remoting MX for Java work?	92
Where does Flash Remoting MX fit into the Java application architecture?	92
Calling Java classes or JavaBeans from ActionScript	94
Making a Java class or JavaBean available to Flash Remoting MX.	94
Getting a reference to a Java class or JavaBean in ActionScript	95
Invoking Java methods in ActionScript.	95
Looking at a Flash application that calls a JavaBean	96
Calling EJBs from Flash.	100
Getting a reference to an EJBHome object in ActionScript	100
Invoking EJB methods in ActionScript.	100
Looking at a Flash application that calls an EJB	101
Calling servlets and JSPs from Flash.	104
Coding a servlet to use with Flash Remoting MX.	104
Getting a reference to a web application in ActionScript.	104
Calling a servlet or JSP	105
Calling JMX MBeans from Flash (JRun only)	107
Getting a reference to an MBean in ActionScript	107
Invoking MBean methods in ActionScript	108
Calling server-side ActionScript from Flash (JRun only)	109
Getting a reference to a server-side ActionScript file.	109
Invoking server-side ActionScript functions	109
Handling function results in ActionScript	111
Using Flash Remoting MX with JRun security	113
Looking at the ActionScript	113
Looking at the JRun security settings	113
Passing XML objects between Flash and Java	114
Sending an ActionScript XML object to Java	114
Returning an XML object from Java to Flash	114
Viewing Flash Remoting MX log entries	115
CHAPTER 7 Using Flash Remoting MX for Microsoft .NET	117
About using Flash Remoting MX for Microsoft .NET.	118
Where does Flash Remoting MX fit into the Microsoft .NET framework?	118
Setting up a Flash Remoting-enabled ASP.NET application.	121
Calling ASP.NET pages from Flash	122
Making an ASP.NET page available to Flash Remoting MX	122
Getting a reference to an ASPX-based service in ActionScript.	122
Invoking ASPX pages in ActionScript.	123
Using the Flash Remoting custom server control in ASPX pages	123
Using the Flash Remoting namespace in code-behind files	124
Using ASP.NET state management with Flash Remoting MX.	125
Using ASP.NET exception handling	127
Using ADO.NET objects with Flash Remoting MX	128
Displaying a RecordSet in Flash with ActionScript.	130

Calling web services from Flash	131
Invoking web service methods using Flash Remoting MX	131
Invoking a remote web service from Flash	132
Calling ASP.NET assemblies from Flash	134
Calling assemblies from Flash	134
Returning an ActionScript object from an assembly	135
Viewing Flash Remoting log entries	137

CHAPTER 8 Flash Remoting ActionScript Dictionary 139

Overview of Flash Remoting ActionScript dictionary	140
ActionScript element documentation conventions	141
Contents of the dictionary	142
DataGlue (object)	143
Method summary for the DataGlue object	143
NetConnection (object)	146
Method summary for the NetConnection object	146
NetDebug (object)	158
Method summary for the NetDebug object	158
NetServices (object)	160
Method summary for the NetServices object	160
RecordSet (object)	164
Method summary for the RecordSet object	164

INDEX 189

ABOUT THIS BOOK

Macromedia Flash Remoting MX provides a communications channel between Macromedia Flash applications and a wide range of business logic and data from ColdFusion, Microsoft® .NET, Java, and Simple Object Access Protocol (SOAP)-based web services. *Using Flash Remoting MX* is intended for Macromedia Flash and application server developers who want to build Rich Internet Applications.

Contents

- Who should read this book..... viii
- Developer resources viii
- About Macromedia Flash Remoting MX documentation..... ix
- Getting answers ix
- Contacting Macromedia ix

Who should read this book

This book is intended for developers with previous experience with Flash and an application server, including Java, Microsoft .NET, or Macromedia ColdFusion MX.

If you are not familiar with Macromedia Flash MX, read the Flash MX documentation set. If you are not familiar with application server development, read the documentation included with your server.

Developer resources

Macromedia, Inc. is committed to setting the standard for customer support in developer education, documentation, technical support, and professional services. The Macromedia website is designed to give you quick access to the entire range of online resources. The following table shows the locations of these resources:

Resource	Description	URL
Macromedia website	General information about Macromedia products and services	http://www.macromedia.com
Information on Flash Remoting MX	Detailed product information on Flash Remoting MX and related topics	http://www.macromedia.com/flashremoting
Macromedia Flash Remoting MX Support Center	Professional support programs that Macromedia offers	http://www.macromedia.com/support/
Flash Remoting MX Online Forum	Access to experienced Flash and application server developers through participation in the Online Forums, where you can post messages and read replies on many subjects relating to Flash Remoting MX	http://webforums.macromedia.com/
Installation Support	Support for installation-related issues for all Macromedia products	http://www.macromedia.com/support/email/isupport
Training	Information about classes, on-site training, and online courses offered by Macromedia	http://www.macromedia.com/support/training
Developer Resources	All the resources that you need to stay on the cutting edge of Flash and application server development, including online discussion groups, Knowledge Base, technical papers, and more	http://www.macromedia.com/desdev/developer/
Macromedia Alliance	Connection with the growing network of solution providers, application developers, resellers, and hosting services creating solutions with Flash Remoting MX.	http://www.macromedia.com/partners/

About Macromedia Flash Remoting MX documentation

The Macromedia Flash Remoting MX documentation is designed to provide support for the complete spectrum of participants. Organized to let you quickly locate the information that you need, the Flash Remoting MX documentation is provided in Adobe Acrobat formats. Flash Remoting MX documentation in Acrobat format is available on the Flash Remoting MX product CD-ROM.

Getting answers

One of the best ways to solve particular programming problems is to tap into the vast expertise of the Flash and application server developer communities on the Macromedia Online Forums. Other developers on the forum can help you figure out how to do just about anything with Flash Remoting MX. The search facility can also help you search messages from the previous 12 months, allowing you to learn how others have solved a problem that you might be facing.

Contacting Macromedia

Corporate
headquarters

Macromedia, Inc.
600 Townsend Street
San Francisco, CA 94103
Tel: 415.252.2000
Fax: 415.626.0554
Web: [http:// www.macromedia.com](http://www.macromedia.com)

Technical support

Macromedia offers a range of telephone and web-based support options. Go to <http://www.macromedia.com/support/flashremoting> for a complete description of technical support services.

You can make postings to the Flash Remoting MX Support Forum (<http://webforums.macromedia.com>) at any time.

Sales

Toll Free: 888.939.2545
Tel: 617.219.2100
Fax: 617.219.2101
E-mail: sales@macromedia.com

CHAPTER 1

Using Flash Remoting MX

Macromedia Flash Remoting MX is an application server gateway that provides a network communications channel between Flash applications and remote services. In this chapter, you learn the basics of Flash Remoting MX, including the Flash Remoting architecture, applying design patterns to Flash Remoting development, and building a simple Flash application with Flash Remoting MX.

Contents

- About Flash Remoting MX..... 2
- Building Flash applications with Flash Remoting MX 5
- Building a Hello World application with Flash Remoting..... 9

About Flash Remoting MX

Flash Remoting MX is an application server gateway that provides a network communications channel between Flash applications and **remote services**. Remote services consist of application server technologies, such as a JavaBean, Macromedia ColdFusion component or page, ASP.NET page, or web service. **Service functions** represent a reference to a specific remote service from ActionScript in a Flash movie.

When compared to other techniques for connecting Flash applications to external data providers, such as HTTP functions like `getURL` and `loadVariables` and XML functions like `XMLSocket`, Flash Remoting MX provides the following advantages:

- **Ease of use** Flash Remoting MX offers automatic data type conversion from native remote service code, such as Java, CFML, and C#, to ActionScript and back again. Also, Flash Remoting MX automatically performs logging, debugging, and security integration.
- **Performance** Flash Remoting MX serializes messages between Flash applications and remote services using the Action Message Format (AMF). AMF is a binary format modeled on the Simple Object Access Protocol (SOAP) format.
- **Extensibility** Flash Remoting MX is designed to integrate with established application design patterns and best practices to build well-designed Flash applications.

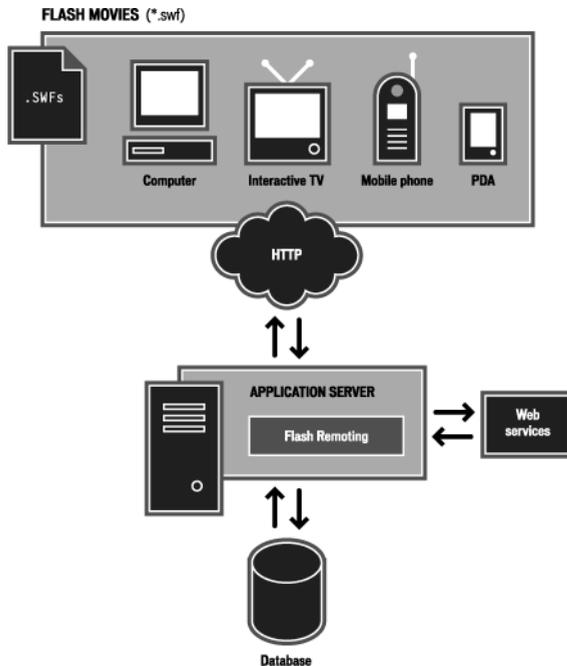
You use the `NetServices` ActionScript functions to connect to application server technologies and web services. In addition, the `NetDebug` and `DataGlue` ActionScript functions help debug Flash applications and display record sets in Flash User Interface (UI) Components.

When compared to traditional HTML-based browser applications, Flash applications provide unique abilities to create dynamic and sophisticated user interactions, including the following:

- Flash Player runtime to execute code, transmit data, and invoke remote services
- Separation of client-side presentation logic from the server-side application logic
- Efficient use of bandwidth by removing the need to refresh the entire page and employing vector-based graphics
- Easy deployment on multiple platforms and devices

On the server side, Flash Remoting MX runs as a servlet in Java application servers, an assembly in .NET servers, and a native service in ColdFusion MX. Depending on the application server platform, Flash Remoting MX on the server contains a series of filters that perform logging, error handling, and security authentication, as well as automatically mapping the service function request to the appropriate server technology.

Using Flash Remoting MX, you can build sophisticated Flash applications, such as a message board, shopping cart, or product catalog. The following figure depicts a simplified representation of the Flash Remoting architecture:



Understanding the Flash Remoting service adapters

Flash Remoting MX automatically maps incoming requests to the appropriate **service adapters**, which provides a direct connection to a specific application server technology. When an HTTP request arrives at the server and contains AMF, Flash Remoting MX maps the request to the appropriate adapter by name. To avoid naming conflicts, you specify the directory structure, fully qualified class or package name, or WSDL URL of the web service in ActionScript.

Understanding AMF

To send and receive messages from remote services, Flash Remoting MX uses Action Message Format (AMF), a binary message format designed for the ActionScript object model. Using AMF, Flash Remoting MX encodes data types back and forth between the Flash application and the remote service over HTTP. Modeled on the Simple Object Access Protocol (SOAP), AMF uses a packet format to relay information. An AMF packet consists of the following parts:

- Packet header that contains AMF version information
- Context header count
- Array of context headers that contain information describing the context in which individual AMF messages should be processed

- Message count
- Array of messages

Server function requests are automatically serialized into AMF format using the NetServices ActionScript functions. On the server, Flash Remoting MX deserializes the incoming AMF messages. When the server-side processing finishes, the results are serialized to AMF and sent back to the Flash application. The format of the server-generated AMF message is identical to the client-generated packet. The body of the individual AMF message contains the error or response object, which is expressed as an ActionScript object.

For more information about error and response objects, see Chapter 2, “Using Flash Remoting Components in ActionScript” on page 13.

Building Flash applications with Flash Remoting MX

Flash applications that use Flash Remoting MX resemble other client-server development platforms, including traditional HTML-based web applications. For example, Flash applications usually appear in the context of a browser window, much like HTML pages. In addition, Flash applications can contain controls for displaying text and graphics, gathering user input, and communicating with a remote server, much like HTML.

Like a web browser request for an HTML page, the Flash application makes a service function call to a remote service. The service function call is a client-initiated, asynchronous event. The Flash application makes a request to the remote service, the service processes the request, and returns the results. The Flash Player does not wait for the result, it handles the result when it is returned.

Understanding the Flash Remoting development environment

Because Flash Remoting MX connects two distinct and separate runtime environments, you build Flash applications with Flash Remoting MX in two programming languages, ActionScript and the programming language of your application server. Therefore, building Flash applications with Flash Remoting MX demands knowledge of at least two different development environments:

- **Flash MX** To create Flash applications that use Flash Remoting MX, you use the Flash MX authoring environment to design the user interface and write the client-side ActionScript.
- **Application server tool** For ColdFusion, Java, or .NET development, you typically use a text editor or an integrated development environment (IDE) that supports the associated programming languages and APIs. Macromedia Dreamweaver MX supports ColdFusion, JSP, and ASP.NET development.
- **Java or .NET compiler** For Java or .NET development, you need a Java or .NET compiler to create executable code.

Because of the separation between the client and server environments, you might develop Flash applications using Flash Remoting MX as a team project. In traditional HTML-based web applications, responsibilities usually fall into two general roles, designer and developer. The designer creates the HTML user interface, and the developer creates the application server logic.

In Flash development using Flash Remoting MX, you might find it useful to organize development roles as server-side developers, client-side developer, and client-side designer. Under this division of labor, the client-side designer creates the Flash user interface, including layout, animation, and effects. The client-side developer creates the ActionScript to connect to the remote service and handle the results. Finally, the server-side developer builds the business logic on the application server to serve as the remote service.

For more information on process planning, see the PetMarket Blueprint application on the Macromedia website at <http://www.macromedia.com/desdev/mx/blueprint/>.

Applying design patterns to Flash Remoting MX

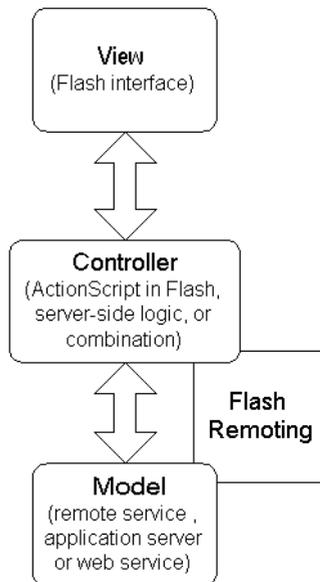
Flash Remoting MX is designed to integrate into established design patterns and frameworks. For Flash applications, Flash Remoting MX streamlines the implementation of properly structured design patterns. Besides increasing development efficiency and reducing mistakes caused by poor designs, patterns can help increase the performance and stability of your applications by forcing you to examine the client-server interactions.

Using the model-view-controller architecture with Flash Remoting MX

Widely adopted by the software development community for user interface-oriented applications, the model-view-controller (MVC) architecture organizes the code of your application by use. The MVC architecture consists of the following elements:

- **Model** The model represents the data of an application and the processing of that data and other logic. In a web application, this typically consists of the application server program and the database.
- **View** The view represents the user interface, which usually consists of user controls and information display.
- **Controller** The controller represents the logic that handles user input and changes the model or view accordingly. Depending on your application design, the controller can be located on the client, the server, or a combination of both. To minimize the amount of network traffic and to take advantage of the Flash runtime, Macromedia recommends implementing the controller in Flash is preferred.

The following figure depicts the MVC architecture in the context of Flash Remoting MX:



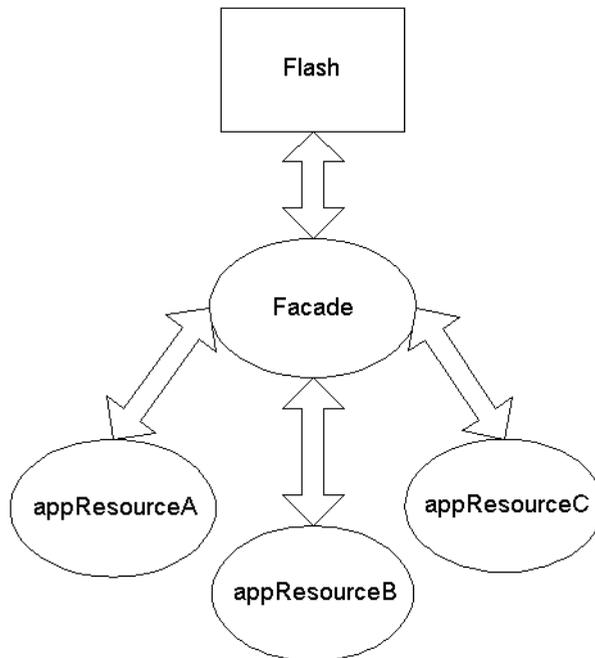
In the figure, Flash Remoting MX enables the separation of the controller from the model by providing a communication channel between Flash applications and application servers or web services. For more information about applying MVC patterns to Flash applications that use Flash Remoting MX, see the PetStore Blueprint application on the Macromedia website at <http://www.macromedia.com/desdev/mx/blueprint>.

Using other design patterns with Flash Remoting MX

Other design patterns exist that try to reduce service function calls, such as the value object pattern. Flash Remoting MX can return results from a remote service as an ActionScript object. An ActionScript object lets you encapsulate service data, which lets you return one object to the Flash application. Instead of numerous individual calls, one object is returned. For more information, see the Value Object entry in the Design Pattern Catalog at http://java.sun.com/blueprints/patterns/j2ee_patterns/value_object/index.html.

To simplify the remote service API available to Flash applications, you can use the facade design pattern to provide a buffer layer between a Flash application and the remote service. The facade pattern prescribes an application server-based broker to receive Flash application service calls and relay the function to the appropriate server resource. This gives you the flexibility of changing your remote service architecture without changing the Flash application.

The following figure depicts the facade design pattern in the context of Flash Remoting MX:



A ColdFusion component, JavaBean, or .NET assembly could serve as the facade by receiving all service function requests from Flash and dispatching the requests to the appropriate application server resource or web service call.

For more information about using the value object and facade design patterns with Flash Remoting MX, see "Software Design Patterns for Flash Remoting" on the Macromedia web site at <http://www.macromedia.com/desdev/articles/facades.html>.

Building a Hello World application with Flash Remoting

In this section, you build a simple Flash application that uses Flash Remoting MX to connect to four different remote services, including a ColdFusion page, a JavaBean, an ASPX page, and a web service. You will see that Flash applications require minimal changes to call different remote services.

To build the Hello World application, see the following sections:

- Building the remote service
- Calling the remote service from ActionScript

Building the remote service

Flash Remoting MX supports Java, ASP.NET, and ColdFusion-based remote services. For a simple Hello World application, the following table lists the application server code by platform and where to save the file to make it available to Flash Remoting MX:

Application Server	Application server code	File location
ColdFusion MX	<pre><cfset flash.result = "Hello from ColdFusion MX!"></pre>	Save the ColdFusion page as helloWorld.cfm in a folder named remoteservices under your webroot.
JRun 4	<pre>package com.remoteservices; import java.io.Serializable; public class FlashJavaBean implements Serializable { private String message; public FlashJavaBean() { message = "Hello from Java!"; } public String helloWorldJava() { return message; } }</pre>	Save the compiled JavaBean class file to classes/com/remoteservices folder under the SERVER-INF directory.
ASP.NET	<pre><%@ Register TagPrefix="Macromedia" Namespace="FlashGateway" Assembly="flashgateway" %> <%@ Page language="c#" debug="true" %> <Macromedia:Flash ID="Flash" runat="server"> Hello from .NET! </Macromedia:Flash></pre>	Save the ASPX page as helloWorldNET.aspx in the flashremoting directory under your webroot.

For more information about application server-specific documentation, see the following chapters:

- Chapter 5, “Using Flash Remoting MX with ColdFusion MX” on page 69
- Chapter 6, “Using Flash Remoting MX for Java” on page 91
- Chapter 7, “Using Flash Remoting MX for Microsoft .NET” on page 117

Calling the remote service from ActionScript

To build a Flash application that uses Flash Remoting MX, you write ActionScript in the Flash MX authoring environment that connects to the remote service and calls a service function. For more information on building Flash applications that use Flash Remoting MX, see Chapter 2, “Using Flash Remoting Components in ActionScript” on page 13.

To build the Flash application that calls remote services using Flash Remoting:

- 1 In the Flash MX authoring environment, create a new Flash file, and save it as `serviceTest fla`.
- 2 In the document window, insert a text field and a PushButton UI component.
- 3 Select the text field with your mouse. In the Properties Inspector, name the text field `messageDisplay`, and select Dynamic Text in the menu.
- 4 Select the PushButton UI component with your mouse. In the Properties Inspector, enter `button_Clicked` in the Click Handler parameter, and enter "Anyone there?" in the Label parameter.
- 5 Open the Actions Panel, and select Actions for Frame 1 of Layer 1 in the menu.

Insert the following ActionScript code:

```
//imports the NetServices ActionScript file
#include "NetServices.as"
//if statement creates the connection to the remote service and creates a
    service object
if (inited == null)
{
    inited = true;
    NetServices.setDefaultGatewayURL("gatewayURL");
    serviceConnection = NetServices.createGatewayConnection();
    serviceObject = serviceConnection.getService("serviceName", this);
}
//function executes when the user clicks the button
function button_Clicked()
{
    //service function call to the remote service
    serviceObject.serviceFunctionName();
}
//if the service function is successful, the _Result function of the same name
    executes
function serviceFunctionName_Result(result)
{
    messageDisplay.text = result;
}
//if the service function is unsuccessful, the _Status function of the same name
    executes
function serviceFunctionName_Status(result)
{
    messageDisplay.text = error.description;
}
```

In the code, the `ActionScript` creates a reference to the remote service using the `NetServices` functions `setDefaultGateway`, `createGatewayConnection`, and `getService`. You specify the following variables:

- The gateway URL depends on the location of the application server running Flash Remoting MX.
- You specify the service function name, which maps to the remote service function name, in the context of the service object.
- You handle the results returned by the service function from the remote service using `serviceName_Result` or `serviceName_Status` functions. If the service function call succeeds, the `_Result` function executes. If the service function fails, the `_Status` function executes.

For more information about connecting to remote services and handling results, see Chapter 2, “Using Flash Remoting Components in ActionScript” on page 13.

6 Save the file.

Using the previous `ActionScript` as a template, the following table lists the values for service name and service function variables:

Remote service	Service name	Service function name
ColdFusion MX	remoteservices	helloWorld
JRun 4	com.remoteservices	helloWorldJava
ASP.NET	remoteservices	helloWorldNET
Web service	http://services.xmethods.net/soap/ urn:xmethods-delayed-quotes.wsdl	getQuote

For more information about application server-specific service name and service function names, see the following chapters:

- Chapter 5, “Using Flash Remoting MX with ColdFusion MX” on page 69
- Chapter 6, “Using Flash Remoting MX for Java” on page 91
- Chapter 7, “Using Flash Remoting MX for Microsoft .NET” on page 117

CHAPTER 2

Using Flash Remoting Components in ActionScript

This chapter describes how to write ActionScript that uses Macromedia Flash Remoting services. It describes how to configure connection information, call service functions, and handle the results. After reading this chapter, you should be able to create a simple application that uses Flash Remoting MX to get data from an application server.

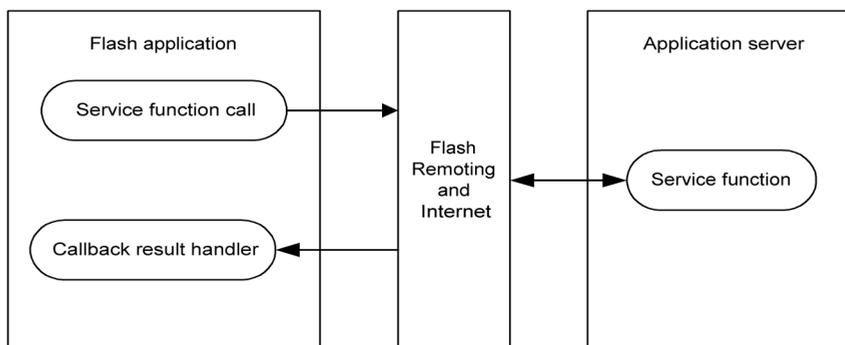
Contents

- Flash Remoting application structure..... 14
- Flash Remoting classes..... 16
- Including Flash Remoting ActionScript files..... 18
- Configuring Flash Remoting MX..... 19
- Calling service functions..... 25
- Handling service results..... 27
- Handling errors..... 31

Flash Remoting application structure

Using Flash Remoting MX to call an application service is similar to calling a web service or making a remote procedure call (RPC), because you make a call to some remote service and you get a response from the service. As with web services and RPCs, data from the remote service gets converted from the native data type of the remote service (such as a Java or C# data type) to a representation that is used to transfer the data over the network.

Unlike an RPC or web service request, a call made using Flash Remoting MX does not directly receive the results of the service. Instead, you write a result-handler callback routine to handle the returned data. Because the remote service call and callback routine are separate, your service call and result handling are asynchronous. In other words, the service request is like a function call without a return, and the service result response is like a Flash event, for which you write an event handler method. The following figure shows this relationship:



To interact with application servers using Macromedia Flash Remoting MX, do the following in your Flash application ActionScript:

- 1 Include the Flash Remoting ActionScript files.
- 2 Configure Flash Remoting MX.
- 3 Call service functions and pass parameters.
- 4 Handle the results and error status information returned to Flash in result event-handler routines.

The following example ActionScript code makes two Flash Remoting service calls, which get a temperature and a forecast. Each Flash Remoting service call has a corresponding result-handler callback method to show the results in the Flash application, and a status handler to handle any error information from Flash Remoting MX.

<p>Include Flash Remoting ActionScript files.</p>	<pre>#include "NetServices.as" // uncomment the next line to use the NetConnection debugger // #include "NetDebug.as"</pre>
<p>Configure Flash Remoting MX.</p>	<pre>if (inited == null) { inited = true; NetServices.setDefaultGatewayURL("http://services. myco.com/flashservices/gateway") gatewayConnection = NetServices.createGatewayConnection(); weatherService = gatewayConnection.getService ("flashExamples.weatherStation", this); }</pre>
<p>Call the service functions.</p>	<pre>weatherService.getTemperature("New York"); weatherService.getForecast("Chicago");</pre>
<p>Callback functions handle returned results and error status information from the service functions.</p>	<pre>function getTemperature_Result(temperature) { temperatureIndicator.text = temperature; } function getForecast_Result(forecast) { forecastIndicator.text = forecast; } function onStatus(errorinfo) { message.text = errorinfo.description; }</pre>

Although a more complex application would be structured differently, the preceding example shows the fundamental elements of a Flash Remoting MX application.

Note: Flash Remoting MX applications must ensure that the r40 revision or later of the Macromedia Flash Player 6 is installed in the client browser. Use the Macromedia Flash deployment kit and the codebase tag to create the detection. The deployment kit is available at http://www.macromedia.com/software/flash/download/deployment_kit/, and includes instructions. For more information on ensuring that the correct version of Flash Player for Flash Remoting MX is installed on the client's system, see the Flash Remoting MX Support Center at <http://www.macromedia.com/support/flash/flashremoting/>.

Flash Remoting classes

The Flash Remoting-related ActionScript classes provide methods to configure Flash Remoting MX, interact with the remote services, and manipulate data on the client. The following figure shows the Flash Remoting-related ActionScript classes and their methods. It divides Flash Remoting classes and methods into three functional areas for configuring Flash Remoting MX and calling services, debugging, and handling record set data, and shows the classes and their methods for each area. (The `NetConnection` class includes methods in two categories.)

<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p style="text-align: center;">NetServices</p> <p>createGatewayConnection() setDefaultGatewayURL()</p> </div> <div style="border: 1px solid black; padding: 5px;"> <p style="text-align: center;">NetConnection</p> <p>getService() setCredentials()</p> <p>addHeader() call() close() connect()</p> </div>	<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p style="text-align: center;">NetDebug</p> <p>trace()</p> </div> <div style="border: 1px solid black; padding: 5px;"> <p style="text-align: center;">NetConnection</p> <p>getDebugConfig() getDebugID() setDebugID() trace()</p> </div>	<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p style="text-align: center;">RecordSet</p> <p>new RecordSet addItem() addItemAt() addView() filter() getColumnNames() getItemAt() getItemID() getLength() getNumberAvailable() isFullyPopulated() isLocal() removeAll() removeItemAt() replaceItemAt() setDeliveryMode() setField() sort() sortItemsBy()</p> </div> <div style="border: 1px solid black; padding: 5px;"> <p style="text-align: center;">DataGlue</p> <p>bindFormatFunction() bindFormatStrings()</p> </div>
<p style="text-align: center;">Configure Flash Remoting and call remote services.</p>	<p style="text-align: center;">Trace events and debug Flash applications.</p>	<p style="text-align: center;">Manipulate and display RecordSet objects returned from remote services.</p>

About the Flash Remoting classes

The following table describes the Flash Remoting classes. For detailed reference information on individual class methods, see Chapter 8, “Flash Remoting ActionScript Dictionary” on page 139.

Class	Purpose
NetServices	<p>Sets a default gateway and establishes a connection (<code>NetConnection</code>) object for the gateway. The gateway is the Flash Remoting MX software that resides on the application server.</p> <p>Use the <code>NetServices.createGatewayConnection</code> method to create a gateway connection (<code>NetConnection</code>) object that connects your Flash application to a Flash Remoting gateway on the application server. For more information on using <code>NetServices</code> methods, see “Configuring Flash Remoting MX” on page 19.</p>
NetConnection	<p>Configures and manages connections to the gateway.</p> <p>The <code>NetConnection.getService</code> method provides access to a service on the gateway. For more information on using this method, see “Creating the service object” on page 22.</p> <p>The <code>NetConnection.setCredentials</code> method provides login information for the gateway server. For more information on using this method, see “Authenticating to the application server” on page 23.</p> <p>The class includes four methods for debugging individual connections. For more information see “Using connection-specific debugging methods” on page 66.</p> <p>The remaining methods in this class (<code>addHeaders</code>, <code>call</code>, <code>close</code>, and <code>connect</code>) are lower-level utilities that are not needed in most applications.</p>
NetDebug	<p>Required by the <code>NetConnection Debugger</code>. Manages the local connection between the Flash application being debugged and the <code>NetConnection Debugger</code>. Includes a single method, <code>NetDebug.trace</code>, which sends a trace event to the <code>NetConnection Debugger</code> with a specific <code>ActionScript</code> object.</p> <p>The <code>NetConnection Debugger</code> reports information on Flash Remoting events. For ColdFusion MX, these events also include server activity, such as SQL queries. For more information on using the <code>NetConnection Debugger</code>, see Chapter 4, “Using the <code>NetConnection Debugger</code>” on page 59.</p>

Class	Purpose
RecordSet	<p>Creates and manipulates RecordSet objects. RecordSet objects typically represent SQL query results and correspond to the ColdFusion Query objects and C# System.Data.DataTable objects. Some remote services return RecordSet objects to Flash.</p> <p>RecordSet methods include functions to create record sets and to add, delete, change, and get record set items. They also can get information about the record set, sort the record set, define an object to receive notifications when the record set changes, and set the way the record set is delivered from the server.</p> <p>For more information on using the RecordSet class, see “Working with RecordSet objects” on page 46.</p>
DataGlue	<p>Binds RecordSet objects to Flash MX components, such as ListBox or ComboBox, that have labels with associated data. The DataGlue.bindFormatStrings and DataGlue.bindFormatFunction methods specify a RecordSet to use to populate the UI component and tell Flash how to populate the component’s labels and data from the RecordSet object.</p> <p>For more information on using DataGlue, see “Using DataGlue methods” on page 56.</p>

Including Flash Remoting ActionScript files

The Flash Remoting ActionScript files contain code that defines many of the Flash Remoting classes. Use the `#include` directives in the following table at the top of the ActionScript in the first frame of the Flash application to include these class definitions in your Flash application. The directives let you use the Flash Remoting classes and methods:

Include statement	Purpose
<code>#include NetServices.as</code>	<p>Enables the NetServices class. All Flash Remoting clients normally use this class. This statement also enables the RecordSet class.</p>
<code>#include NetDebug.as</code>	<p>Enables NetConnection Debugger output and the NetDebugConfig and NetDebug classes. Include this statement during development to enable debugging of Flash Remoting applications.</p> <p>You must remove this line before you publish your Flash application for deployment to a production server. By removing this statement, you prevent access to debugging information from remote sites.</p>
<code>#include DataGlue.as</code>	<p>Enables the DataGlue class. Use this statement if you are using DataGlue methods to simplify providing RecordSet data to Flash MX UI components.</p>

Configuring Flash Remoting MX

Before you can call a remote service, you must configure Flash Remoting MX:

Note: The client does not access the remote server when you configure Flash Remoting MX. The client first communicates with the remote server when it makes a first service function call.

To configure Flash Remoting MX:

- 1 Create the Flash Remoting connection object.
- 2 Create the service object.
- 3 If your application server or service requires user authentication, provide the authentication information.

The following sections describe these steps.

Creating the Flash Remoting connection object

When you configure a Flash Remoting connection, you identify a remote gateway for Flash to use in remote service requests and create a `NetConnection` object that enables Flash Remoting MX to connect to the remote gateway.

To create a Flash Remoting connection object:

- 1 Specify the gateway URL.
- 2 Create a `NetConnection` object that uses the specified URL.

You have several options for doing these steps, including the following:

- You can use the `NetServices.createGatewayConnection` method to specify the URL and create the `NetConnection` object.
- You can use a `NetServices.setDefaultGatewayURL` method to specify a default gateway, and use the `NetServices.createGatewayConnection` method without a URL to create the `NetConnection` object. Flash Remoting MX uses the default gateway for all connections that you create without otherwise specifying a gateway.
- You can specify the gateway in the web page you use to deploy the SWF file, and use the `NetServices.createGatewayConnection` method without a URL to create the `NetConnection` object.

You can combine these techniques. If you do so, the gateway URL is determined as follows:

- 1 A URL specified in a `createGatewayConnection` method takes precedence over any other URL.
- 2 A URL specified in the deployed web page takes precedence over a default gateway URL.
- 3 If you do not otherwise specify a gateway URL, Flash Remoting MX uses the gateway specified in a `NetServices.setDefaultGatewayURL` method.

Macromedia recommends using the `setDefaultGatewayURL` method to specify the URL of the gateway that you use in the Flash MX authoring environment. Doing so lets you test your Flash Remoting application directly in the development environment, without having to change your code when you deploy your application. When you deploy the SWF file, supply the production gateway URL in the web page that you use to run the Flash application. This way, you also do not have to make any changes in your Flash application to deploy the SWF file on different application servers.

Specifying the gateway in the `NetServices.createGatewayConnection` method

To configure a connection to a specific gateway, use the `createGatewayConnection` method; for example:

```
var gatewayConnection = NetServices.createGatewayConnection("http://apps.  
mycompany.com/flashservices/gateway");
```

The `gatewayConnection` object returned by this method is the `NetConnection` object for the connection. You use this object to set security credentials, if required, and to get the Flash Remoting service or services that you will use.

This technique has the disadvantage of a hard-coded URL. To change the gateway URL, you must change the ActionScript and publish and deploy your movie.

Using the `NetServices.setDefaultGatewayURL` method

The `setDefaultGatewayURL` method sets a default value for the gateway URL, which the `createGatewayConnection` method uses if you do not otherwise supply the Flash Remoting service URL. For example:

```
NetServices.setDefaultGatewayURL("http://apps.mycompany.com/flashservices/gateway")  
var gatewayConnection = NetServices.createGatewayConnection();
```

Because you set a default gateway URL, you do not always need to specify the gateway URL elsewhere. However, you can override the default value by specifying a gateway URL in the `createGatewayConnection` method or in the web page that calls your SWF file.

Setting a default gateway URL provides you with the greatest flexibility in both development and deployment. It lets you provide a URL that works when you test your movie directly in the Flash development environment. It also lets you override the default value with a server-specific gateway when you deploy your movie.

Note: If you specify `localhost` as the host in the `setDefaultGatewayURL` method, and you run your Flash application from outside the Flash development environment or stand-alone Flash player, Flash Remoting MX does not use `localhost` as the default gateway host. Instead, it replaces `localhost` and any port specified in the `setDefaultGatewayURL` method with the host and port specified to run the Flash application. For example, if you specify `http://localhost/flashservices/gateway` in the `setDefaultGatewayURL` method and start your Flash application by using the URL `http://apps.mycompany.com:8500/flashapps/myapp.swf` in a web page or browser, Flash Remoting MX uses `http://apps.mycompany.com:8500/flashservices/gateway` as the default gateway URL.

Specifying the gateway in a web page

You specify the gateway in the web page that loads your SWF file by adding entries in the OBJECT tag that calls the SWF file. You must use different techniques for Internet Explorer and Netscape browsers:

- To specify the gateway for Internet Explorer, specify the gateway URL as a `flashvars` parameter in an HTML PARAM tag inside the OBJECT tag body.
- To specify the gateway for Netscape, specify the gateway URL as a `flashvars` attribute to the HTML EMBED tag that specifies the Flash application.

The following HTML example runs the `myMovie.swf` Flash application and specifies `http://apps.myCompany.com/flashservices/gateway` as the URL for the Flash Remoting services gateway. The first highlighted line contains the code for Internet Explorer. The second highlighted line contains the code for Netscape.

```
<OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
  codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.
  cab#version=6,0,0,0"
  WIDTH="100%"
  HEIGHT="100%"
  id="MyMovie">
  <PARAM NAME="flashvars" VALUE="gatewayURL=http://apps.myCompany.com/
  flashservices/gateway">
<PARAM NAME=movie VALUE="MyMovie.swf">
<PARAM NAME=quality VALUE="high">
<PARAM NAME=bgcolor VALUE="#000099">
  <EMBED src="MyMovie.swf"
    FLASHVARS="gatewayURL=http://apps.mycompany.com/flashservices/gateway"
    quality=high bgcolor="#000099"
    WIDTH="100%"
    HEIGHT="100%"
    NAME="movieName"
    TYPE="application/x-shockwave-flash"
    PLUGINSPAGE="http://www.macromedia.com/go/getflashplayer">
  </EMBED>
</OBJECT>
```

Note: If you specify the Flash Remoting gateway in the web page, the `ActionScript #include NetServices.as` directive must be on the main movie timeline, not in a movie clip.

Identifying the gateway

The specific format of the URL that you use to specify the gateway depends on the application server to which you are connecting:

- If you are connecting to a Java server or Macromedia ColdFusion MX, use the following format:
`http://webServer[:port]/flashservices/gateway`
In this URL, `flashservices` is the name of the Java application context, and `gateway` is the servlet mapping. If you do not use the default Flash Remoting deployment configuration, replace `flashservices` with the application context, and replace `gateway` with the servlet mapping.
- If you are connecting to a .NET server, use the following format:
`http://webServer[:port]/flashremoting/gateway.aspx`

In this URL, `flashremoting` is the logical directory used for Flash Remoting MX, and `gateway.aspx` is an intentionally blank file installed with Flash Remoting MX; the `aspx` suffix identifies this as a .NET request. If you do not use the standard installation configuration for Flash Remoting MX, you might need to change these values to reflect your configuration.

Creating the service object

Before you access a service function, you must use the `getService` method of the gateway connection object to create a service object in the Flash client. The `gatewayConnection.getService` method takes the following two parameters:

- The service name.
- Optionally, the name of the Flash responder object that will receive the results. If you omit the Flash responder, you must specify a Flash responder in each service function call. For more information on considerations for specifying a Flash responder, see “Handling service results” on page 27.

The following example shows the `getService` method for a weather service:

```
if (inited == null)
{
    inited = true;
    NetServices.setDefaultGatewayURL("http://localhost/flashservices/gateway")
    gatewayConnection = NetServices.createGatewayConnection();
    weatherService =
        gatewayConnection.getService("flashExamples.weatherStation", this);
}
```

In this example, you create the `weatherService` service object.

By specifying `this` as the responder object, you specify that the current Flash object is also the responder. It is a common practice to create the service object and define the result-handler callback routines in a single Flash object.

Specifying a service

The service name format depends on the type of service you are using. The following table lists how you specify service names for the supported service types:

Service type	Service name	Example
Web services (accessed through ColdFusion or .NET servers)	URL of the WSDL file for the service.	<code>http://www.xmethods.net/sd/2001/BabelFishService.wsdl</code>
ASP.NET pages (.aspx)	Logical directory path from the web root. Replace all slash or backslash characters in the path with periods.	<code>FlashGateway.Samples.WeatherService</code>
DLL files (.dll)	Fully qualified class name.	<code>Flashgateway.Samples.WeatherService</code>
EJBs	JNDI name of the EJBHome binding.	<code>com.mycompany.samples.WeatherService</code>
Java classes, including JavaBeans	Fully qualified Java class name.	<code>com.mycompany.samples.WeatherService</code>

Service type	Service name	Example
Java servlets	Web application context root.	WeatherService
JMX (JRun 4 only)	MBean object name.	DefaultDomain:service=DeployerService
ColdFusion pages (.cfm)	Path from the web root to the page's directory. Replace all forward slash or backward slash characters in the path with periods.	Flashgateway.samples.WeatherService
ColdFusion components (.cfc)	Qualified ColdFusion component name starting from the web root.	Flashgateway.samples.WeatherService

For example, to create a service object for the BabelFish web service, specify a line similar to the following:

```
babelFishService = gatewayConnection.getService("http://www.xmethods.net/sd/2001/
BabelFishService.wsdl", this);
```

EJB considerations

If you are accessing EJBs directly from `ActionScript`, the `gatewayConnection.getService` method returns the Home interface of the EJB. You must then use the Home interface `create` method to access the EJB. For more information, see “Calling EJBs from Flash” on page 100.

Authenticating to the application server

In some cases, your application server might require you to provide user authentication—a user name and password—before you can use a service.

Flash Remoting MX provides the `setCredentials` method of the `NetConnection` object to specify authentication credentials to send to the application server. Flash then sends the credentials in the header of every service function call. This method is currently supported on the following servers:

- ColdFusion MX
- JRun 4

To use the `setCredentials` method, call the method after you have created the gateway object and before you call a service method. For a Flash application that is run by a single user during a session, you can call the `setCredentials` method any time after creating the gateway connection object, as follows:

```
#include "NetServices.as"
NetServices.setDefaultGatewayURL("http://www.mySite.com/flashservices/gateway");
gatewayConnection = NetServices.createGatewayConnection();
gatewayConnection.setCredentials(username, password)
serviceObject = gatewayConnection.getService("myService", this);
```

Note: To ensure security, never include specific user names or passwords in `ActionScript`. Flash supplies the login credentials with each service request; therefore, your `ActionScript` should log the user out of the application server *and* reset the credentials when the user logs out of your Flash application.

To log out the user and reset the credentials:

- 1 Call a logout service method on the server that logs out the user. (For example, for ColdFusion, call a method that uses the `cflogout` tag.)
- 2 Set the gateway connection credential information to empty strings.

The following example shows these steps:

```
myService.logout();  
gatewayConnection.setCredentials("", "");
```

The technique you use on the application server to authenticate the user and authorize access depends on the application server.

Using authentication in ColdFusion MX

If you use ColdFusion MX, the `setCredentials` method causes Flash Remoting MX to put the login ID and password in the ColdFusion page's `CFLogin` scope whenever you request a service. The `Application.cfm` page for the ColdFusion page or ColdFusion component should process this information as necessary in a `cflogin` tag. For more information on implementing security in ColdFusion, see “Securing access to ColdFusion from Flash Remoting MX” on page 88.

Using authentication in JRun 4

Authentication in JRun 4 is only meaningful for EJBs, where you can use the `setCredentials` method to enable access to secured EJB methods. You must configure your application server for single sign on. The Flash gateway provides the security credentials to the login module. For more information on authentication in JRun 4, see “Using Flash Remoting MX with JRun security” on page 113.

Calling service functions

To call the functions exposed by a service object, you use the service object name followed by the application server functionality name. The following is an example:

```
weatherSvc.getTemperature("New York");
```

In this example, the `getTemperature` function exists in the application server as a public method or application page. The function also passes a string parameter, `New York`. To pass multiple parameters to service functions, you include a comma-separated list of values in the service function call; for example:

```
weatherSvc.getTemperature("New York", 1998, "average");
```

In this example, the service function passes three parameters, a city (`New York`), a year (`1998`), and a command (`average`).

Note: Parameters must be in the order required by the service function.

Specifying a responder object

If you do not specify a result handler when you create the service object, you must specify a result-handler callback object for the service function when you call it. If you do specify a result handler when you create the service object, you must not specify a result-handler callback object for the service when you call it. Therefore, you must do either of the following:

- Use a single object for all of a service's methods.
- Specify a responder for each method.

Note: Do not specify a result handler both when you create the service object and when you call a service method. If you specify the result handler in both places, Flash Remoting MX passes an object representing the handler as the first argument to the service function. This will cause errors in your application that might be difficult to diagnose.

To specify a result handler for a specific service function call, add the result handler as the first entry in the function argument list, as in the following example:

```
weatherSvc.getTemperature( new temperatureResult(), "New York", 1998, "average" );
```

In this case, the `temperatureResult` result handler will receive and process the results, including any error status information, of the `getTemperature` service function.

For information on creating and using function-specific result handlers, see “Handling service results” on page 27.

Calling functions using named arguments in ColdFusion

If you are calling a ColdFusion page or ColdFusion component that can take named arguments, you can also call a service function and pass it a single `ActionScript` object that contains name-value pairs for the arguments, as shown in the following example:

```
myService.myFunction( { dept: "Sales", name: "BobZ" } );
```

The ColdFusion page or component uses the object elements, `dept` and `name`, as named arguments. For more information on passing parameters to ColdFusion, see “Passing parameters to ColdFusion components” on page 78.

Specifying functions

The way you specify the service function name depends on the type of service you are using. The following table lists how you specify the function names for the supported service types:

Service type	Function name
Web services (SOAP-based)	Web service method exposed through WSDL
ASP.NET pages (.aspx)	ASP.NET page (without a suffix)
DLL files (.dll)	Public method
EJBs	EJBHome and EJBObject method
Java classes, including Java Beans	Public method
Java servlets	Servlet-name registered in the web.xml file
JMX (JRun 4 only)	MBean method
ColdFusion pages (.cfm)	ColdFusion page (without the cfm suffix)
ColdFusion components (.cfc)	Component method

Note: Calling web services using Flash Remoting MX is only supported in Flash Remoting MX for .Net and in the Flash Remoting MX support included with ColdFusion MX. Flash Remoting MX for Java and the Flash Remoting MX support in JRun do not support calling web services using Flash Remoting MX.

Handling service results

When the service function results return from the application server, event handlers use the returned data or handle the returned error information. For example, a result handler could display the results in the Flash application, and an error handler could set trace functions to report the error descriptions. This section describes how to handle result data. The section “Handling errors” on page 31 describes how to handle error information.

Result-handling hierarchy

Flash Remoting MX supports the following hierarchy of event handling:

- 1 If you specify a responder object in the `NetConnection` object `getService` method, Flash Remoting MX does the following:
 - a If the responder has a function with a name of the form *functionName_Result*, where *functionName* is the name of the service function that you called, Flash Remoting MX returns the result for the function to that method.
 - b If the responder has a function named `onResult`, Flash Remoting MX returns the result for the function to that function.
- 2 If you specify a responder object in the service function call, Flash Remoting MX returns the result to that object's `onResult` method.

Note: Do not specify a responder object in the `gatewayConnection.getService` method and in the service function call. If you specify the responder in both places, Flash Remoting MX passes the responder you specify in the service function method to the service function as an argument. This behavior causes errors in your application.

- 3 When you test an application in Flash MX during development, if there is no appropriate responder object method, Flash displays the results in a message window.

Result-handling strategies

The different types of result handlers provide you with a variety of result-handling strategies you can use to match your application's needs. The following sections describe some of the techniques you can use:

- “Using the `getService` method to specify a responder”
- “Using the service function call to specify a responder”

Using the `getService` method to specify a responder

If you use the `gatewayConnection.getService` method to specify the responder object, you can use the following techniques:

- Specify an object, typically the main movie `this` object, as the responder in the `gatewayConnection.getService` method and create a separate `_Result` handler on the main movie timeline for each service function. Use this technique if all of the following conditions are true:
 - The results of calls to one function name must be handled differently from the results of calls to another function.
 - All functions in all services that require different handling have unique names. (For example, you do not use two services with identically named functions, such as `myFirstService.myFunction` and `myOtherService.myFunction`, that require different processing.)
 - The results of all calls to a particular function in a service can be handled by a single responder.
- Specify the main movie `this` object as the responder in the `gatewayConnection.getService` method and create an `onResult` method on the main movie timeline to handle some or all of the service function results. Use this technique if the results of all requests for all service functions can be handled in the same manner.
- Use a combination of the previous two techniques. Specify the main movie `this` object as the responder in the `gatewayConnection.getService` method, and create a separate `_Result` handler for some service functions. Create a single `onResult` result handler for the remaining service functions. Use this technique if the following are true:
 - The results of some service functions must be handled differently from the results of other service functions.
 - All functions in all services that require different handling have unique names.
 - The results of several service functions with different names can be handled in the same manner.
 - The results of all calls to a particular function can be handled by a single responder.

For an example of this technique, see the following section, “Example: using a hierarchy of handlers”.

Example: using a hierarchy of handlers

The following example shows how you can use the main movie as the responder and have a common `onResult` result handler for some service functions, and function-specific `functionName_Result` handlers for other functions. In this example, there are two function-specific result handlers, `getTemperature_Result` and `getForecast_Result`, which display the returned temperature and forecast in specific text boxes. The default `onResult` response handler displays the result in a general-purpose message box.

```
// Initialization code, run once for each movie instance.
if (inited == null)
{
    inited = true;
    NetServices.setDefaultGatewayURL("http://localhost/flashservices/gateway")
    gatewayConnection = NetServices.createGatewayConnection();
    // Specify the main movie (this) as the default responder object.
    weatherService = gatewayConnection.getService("flashExamples.weatherStation",
        this);
}

// Function-specific result handlers
function getTemperature_Result(temperature)
    { temperatureIndicator.text = temperature; }
function getForecast_Result(forecast)
    { forecastIndicator.text = forecast; }
// Default response handler
function onResult(result)
    { generalMessageBox.text = result; }

// Call the service functions
weatherService.getTemperature("New York");
weatherService.getForecast("Chicago");
weatherService.getServiceStatus("San Francisco");
weatherService.getUsageStats();
```

Using the service function call to specify a responder

If you specify the responder when you call the service function methods, you can use the following techniques:

- Define a responder object. Populate the responder object with result handler functions, as described in any of the bullets in “Using the `getService` method to specify a responder” on page 28. Pass a new instance of the responder object to the `gatewayConnection.getService` method.
This technique is more object-oriented than using the `this` object as a responder, and lets you program using more encapsulated code than the techniques in “Using the `getService` method to specify a responder” on page 28.
- Define several unique responder objects. Populate the responder objects with result-handler functions, as described in any of the bullets in “Using the `getService` method to specify a responder” on page 28. Pass a new instance of one of these responder objects to each `serviceName.functionName` call you make.
This technique lets you create function-specific responders where several services have the same function names but return different data. It also lets you create several

different handlers that you can use, under different circumstances, with a single service function.

For an example of this technique, see “Example: specifying unique result objects in service function calls”, next.

- Use a combination of the previous two methods: use some responder objects for multiple service functions and use unique responder objects for other service functions.

Example: specifying unique result objects in service function calls

The following example rewrites the example in the “Example: using a hierarchy of handlers” on page 29 to have the service function calls specify result handlers. There are three response-handler objects: `tempResult` for the temperature, `forecastResult` for the forecast, and `generalResult` for other service functions. Each result handler has one function, `onResult`, to handle the result returned by the service.

```
// Initialization code, run once for each movie instance.
if (inited == null)
{
    inited = true;
    NetServices.setDefaultGatewayURL("http://localhost/flashservices/gateway")
    gatewayConnection = NetServices.createGatewayConnection();
    // Do not specify a default responder object when creating the service object.
    weatherService = gatewayConnection.getService("flashExamples.weatherStation");
}

// Temperature result handler object
function tempResult()
{
    this.onResult = function (temperature)
    { temperatureIndicator.text = temperature; }
}

//Forecast result handler object
function forecastResult()
{
    this.onResult = function (forecast)
    { forecastIndicator.text = forecast; }
}

// General result handler object
function generalResult ()
{
    this.onResult = function (result)
    { generalMessageBox.text = result; }
}

// Call the service functions and specify the result handler as the first argument.
weatherService.getTemperature(new tempResult(), "New York");
weatherService.getForecast(new forecastResult(), "Chicago");
weatherService.getServiceStatus(new generalResult(), "San Francisco");
weatherService.getUsageStats(new generalResult());
```

Handling errors

Flash Remoting MX returns a Status event instead of a result if either of the following happens:

- The Flash Remoting gateway encounters an error.
- A service function encounters an error (throws an exception).

You write status event handlers to respond to the errors, typically by displaying an error message or logging the error information. In some cases, you might be able to include recovery code, such as code to retry a call to a busy server, in the error handler.

The error object

When Flash Remoting MX receives a Status event, Flash passes an error object that contains information about the error to the status event handler. The error object has the following format:

Key name	Contents
code	Currently, always "SERVER.PROCESSING".
level	Currently, always "Error".
description	A string that describes the error.
details	A stack trace that indicates the processing state at the time of the exception.
type	The error class name.
rootcause	A nested error object that contains additional information on the cause of the error. Provided only if a Java ServletException is thrown.

Error-handling hierarchy

Flash Remoting MX supports the following hierarchy of error handling:

- 1 The initial error responder depends on whether you have specified the responder in the service function or the `getService` call:
 - If you specify the responder in the `getService` call, the following happens:
 - a. If the responder has a function with a name of the form `functionName_Status`, where `functionName` is the name of the service function that you called, Flash Remoting MX returns the status information for the function to that method.
 - b. If the default responder object specified in the `getService` call has a function named `onStatus`, Flash Remoting MX returns the status to that function.
 - If you specify a responder object in the service function call, Flash Remoting MX returns the status to that object's `onStatus` method. When you use this technique, the responder object must also have an `onResult` method to handle the error.
- 2 If there is a function called `_root.onStatus`, Flash Remoting MX returns the status to that function.

- 3 If there is a function called `_global.System.onStatus`, Flash Remoting MX returns the status to that function.
 - 4 During development, Flash displays the status information in a message window.
- This hierarchy adds two levels, items 2 and 3, to the result-handling scheme described in “Result-handling hierarchy” on page 27. As a result, you can define handlers for all otherwise-unhandled status events on a level or in the entire application.

Error-handling strategies

Flash applications use error-handling code less often than server applications, because Flash does not report errors to the user when viewing movies. However, Flash Remoting MX does make error information available to the movie ActionScript, and your application can use this information. The error handling hierarchy lets you handle errors with any degree of granularity. For example:

- You can use a global error handler to log error information if the status event cannot be otherwise handled.
- You can display a message to the user in response to specific functions or specific errors returned by specific functions.

Error handling can be particularly useful for remote services, because the user’s experience depends on remote data that might not always be retrieved, and it might not be obvious that data is not retrieved. Also, in some cases, the error-handling code can even recover from a transient error. For example, if a service function fails due to a time-out, it might be appropriate for the Flash application to try the request a second time. If the request fails a second time, the movie could then display a message to the user and post a message to the server to log the error.

The following section, “Example: error handling using unique result objects” shows how you can apply error handling to a Flash Remoting MX application. For more information about selecting from the status event-handling hierarchy, see “Result-handling strategies” on page 27, which discusses strategies for selecting from the similar result-handling hierarchy.

Example: error handling using unique result objects

The following example adds error handling to the example in the “Example: specifying unique result objects in service function calls” on page 30. The error handlers for the temperature and forecast display custom messages to the user. The general error handler displays a generalized message. All handlers call a function to report the error to a log on the server.

```
#include "NetServices.as"
#include "NetDebug.as"

// Initialization code, run once for each movie instance.
if (inited == null)
{
    inited = true;
    NetServices.setDefaultGatewayURL("http://apps.myco.com/flasheservices/gateway")
    gatewayConnection = NetServices.createGatewayConnection();
    // Do not specify a default responder object when creating the service object.
```

```

    weatherService = gatewayConnection.getService("flashExamples.weatherStation");
}

// Temperature result handler object
function tempResult()
{
    this.onResult = function(temperature)
    {
        temperatureIndicator.text = temperature;
    }

    this.onStatus = function (status)
    {
        temperatureIndicator.text = "No Temperature Available";
        _global.logStatus(Status);
    }
}

//Forecast result handler object
function forecastResult()
{
    this.onResult = function (forecast)
    { forecastIndicator.text = forecast; }

    this.onStatus = function (status)
    {
        forecastIndicator.text = "No Forecast Available";
        _global.logStatus(Status);
    }
}

// General result handler object
function generalResult ()
{
    this.onResult = function (result)
    { generalMessageBox.text = result; }

    this.onStatus = function (status)
    {
        generalMessageBox.text = "An error occurred. Please try later";
        _global.logStatus(Status);
    }
}

// Call the service functions and specify the result handler as the first argument.
// In a real application, these calls would be initiated by user actions in the
// Flash application.
weatherService.getTemperature(new tempResult(), "New York" );
weatherService.getForecast(new forecastResult(), "Chicago" );
weatherService.getServiceStatus( new generalResult(), "San Francisco" );
weatherService.getUsageStats(new generalResult());

```


CHAPTER 3

Using Flash Remoting Data in ActionScript

This chapter describes how to handle data that your ActionScript send a to and receives from service functions. It discusses how Macromedia Flash Remoting MX converts data types, type-specific data handling issues, and how to use simple and complex data types. It also discusses using `RecordSet` objects and other object types in detail.

Contents

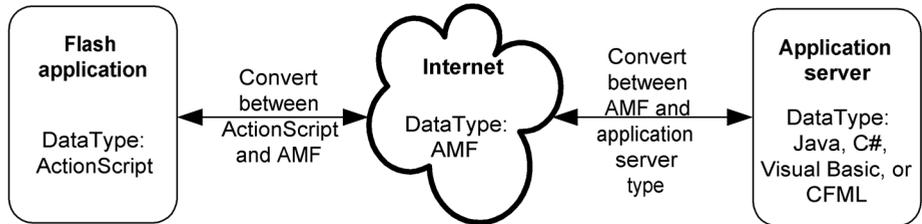
- About Flash Remoting MX and data types..... 36
- Converting from ActionScript to application server data types..... 37
- Converting from application server data types to ActionScript..... 39
- Working with objects..... 43
- Working with `RecordSet` objects 46
- Working with XML..... 57

About Flash Remoting MX and data types

When Flash Remoting MX sends data from a Flash application to an application server, and when a server returns data to a flash application, the data is converted twice:

- In the Flash client, between ActionScript data types and Action Message Format (AMF)
- At the server gateway, between Action Message Format and the native language of the application (Java, C#, Visual Basic, or CFML).

The following figure shows this data conversion:



A two-step conversion process enables Flash Remoting MX to use a system-neutral, efficient representation for the data it transmits. This way, you can create a Flash application that can work with many back-end application servers.

Flash Remoting MX does all data conversions automatically. In most cases, the conversion is straightforward; you simply pass the ActionScript data as arguments to a remote service, and handle the returned data in your callback functions. In a few cases, you must understand data type conversion considerations to ensure correct application behavior. This chapter includes information about these considerations.

Converting from ActionScript to application server data types

Flash Remoting MX automatically converts between ActionScript data and the data types specific to the application server programming environment.

In strictly typed environments, such as Java and C#, the service function data types must correspond to (or be derived from) the listed data types. In loosely typed programming environments, such as ColdFusion or Visual Basic, the environment must be able to use the listed data type for the service function variable.

The following table lists the server data types into which the Flash gateway converts ActionScript data types:

ActionScript	Java	C#	Visual Basic	SOAP	ColdFusion MX
null	null	null	Nothing	null	null (not defined) ColdFusion uses a null value internally, but reports an undefined variable
undefined	null	null	Nothing	null	null (not defined)
Boolean	Boolean	System.Boolean	Boolean	boolean	Boolean
Number	Number	Any numeric type, as appropriate	Any numeric type, as appropriate	decimal, float, double, integer, int	Number
String	String	System.String	String	string	String
Date	Date	System.DateTime	Date	dateTime	Date
Array (contiguous numeric indexes)	ArrayList	System.Collections.ArrayList	System.Collections.ArrayList	array	Array
Associative Array (named indexes)	java.util.Map (case-insensitive implementation)	System.Collections.Hashtable	System.Collections.Hashtable	Complex type	Struct
RecordSet	Cannot be sent to Server	Cannot be sent to Server	Cannot be sent to Server	Cannot be sent to Server	Cannot be sent to Server
Object	flashgateway.io.ASObject (which implements java.util.Map)	FlashGateway.IO.ASObject (which implements the ICollection interface)	FlashGateway.IO.ASObject	Complex type	Struct Object of type FlashGateway.IO.ASObject For more information, see "Working with ActionScript typed objects" on page 43

ActionScript	Java	C#	Visual Basic	SOAP	ColdFusion MX
Object consisting of name-value pairs passed as the only argument to a service function (Typically used to pass data to ColdFusion)	flashgateway.io.ASObject	FlashGateway.IO.ASObject	FlashGateway.IO.ASObject	Complex type	For a CFML page – the Flash Scope with the object key names being the scope variables For a ColdFusion component (CFC) – Each name-value pair is a named argument to the function For more information, see “Accessing ActionScript objects” on page 73
XML	org.w3c.dom.Document	System.Xml.XmlDocument	System.Xml.XmlDocument	Cannot be converted	XML

ActionScript data conversion notes

The following is additional information on conversion from ActionScript data types to server data types:

- Flash Remoting MX converts ActionScript Numbers to any valid Java or .NET numeric data type wherever possible. If the number cannot be converted to the service data type, for example, if the number exceeds the valid range of an integer data type, Flash Remoting MX throws an error on the server.
- Flash Remoting MX can convert ActionScript data to Java boolean and numeric primitive types that correspond to the listed class types.
- Flash Remoting MX treats all ActionScript arrays with non-contiguous indexes (for example, arrays with indexes of 0, 2, and 3 but not 1) or with both numeric and text indexes (for example, arrays with indexes of 0, 1, 2, “data1”, and “data2”) as associative arrays. In these cases, it converts the numeric indexes to keys with numbers as the strings (for example, “0”, “1”, “2”).

Converting from application server data types to ActionScript

Flash Remoting MX automatically converts between the data types specific to the application server programming environment and ActionScript.

The following table shows how Flash Remoting MX converts data returned from the application server to ActionScript data types:

C#	Visual Basic	SOAP	ColdFusion MX	Java	ActionScript
null	Nothing	null	N/A	null	null
bool System.Boolean	Boolean	boolean	Boolean, 0, or 1 For details, see "Boolean data in ColdFusion" on page 41	Boolean	Boolean
any number type	any number type	decimal, float, double	Val(Number) For details, see "Numeric data in ColdFusion" on page 41	Number	Number
System.Char System.String	Char String	string	String	String Character	String See "Server data conversion notes"
System.DateTime	Date	dateTime	Date	Date	Date
System.Collections. ICollection object[]	System.Collections. ICollection object[]	array	Array	Collection Object[] array of primitive types	Array
System.Collections. Hashtable System.Collections. IDictionary	System.Collections. Hashtable System.Collections. IDictionary	complex type	Struct	java.util.Map	Associative array
Sytem.Data. DataSet	Sytem.Data. DataSet	Complex type (DataSet)	-	-	Associative array of RecordSet objects
System.Data.Data Table	System.Data.Data Table	-	Query object	java.sql.Result Set	RecordSet
-	-	-	Query object (Flash.pagesize variable set)	flashgateway. sql.Pageable ResultSet	Paged RecordSet

C#	Visual Basic	SOAP	ColdFusion MX	Java	ActionScript
FlashGateway.IO. ASObject System.Exception	FlashGateway.IO. ASObject System.Exception	Complex type	A Java object as listed in the next column	flashgateway.io. ASObject Serializable Dictionary Throwable	Object
FlashGateway.IO. ASObject with Type property set	FlashGateway.IO. ASObject with Type property set	Complex type	Java Object of class flashgateway.io. ASObject with Type property set (Set the type using a Java expression once you create the object using CFML.)	flashgateway.io. ASObject with Type property set	Typed Object See “Server data conversion notes”
System.Xml. XmlDocument	System.Xml. XmlDocument	–	XML	org.w3c.dom. Document flashgateway.io. ASXMLString	XML

Server data conversion notes

The following is additional information on conversion from server data types to ActionScript data types:

- If a string data type on the server represents a valid number in ActionScript, Flash can automatically cast it to a number if needed.
- If you use the setType method to assign an object type to a flashgateway.io.ASObject object on the server, and the type name matches the name of a registered class in ActionScript, Flash Remoting MX creates an instance of that type in ActionScript. For more information see “Working with ActionScript typed objects” on page 43.
- To return multiple, independent, values to your Flash application, place them in a complex server variable that can hold all the required data, such as a variable that converts to a Flash Object, Array, or Associative Array type. Return the single variable and use its elements in the Flash application.

ColdFusion to ActionScript data conversion issues

ColdFusion is a loosely typed or “untyped” language, where the data type of a variable can be ambiguous. As a result, Flash Remoting MX cannot always determine how to convert between ColdFusion data and ActionScript data. The following sections discuss the limitations that this situation imposes, and how to prevent errors that can arise as a result.

Boolean data in ColdFusion

If a ColdFusion page or CFC returns Boolean values, it represents these values as strings. Flash does not have rules for converting strings to Boolean values. Instead, it converts the string to a number, and then converts the number to a Boolean value. This operation converts all representations of Boolean values except 1 to False. Therefore Flash converts ColdFusion Boolean values of “Yes”, “True”, and True to False.

To return a Boolean value correctly from ColdFusion to ActionScript, do either of the following:

- Return a 1 (True) or 0 (False) numeric or string value. For example, the following function converts any ColdFusion Boolean value to value that ActionScript can use correctly. (For simplicity, this example omits error-handling code.) Your ColdFusion page can call this function before returning a Boolean value to Flash Remoting MX:

```
<cffunction name="convertBool">
    <cfif Arguments[1] >
        <cfreturn "1">
    <cfelse>
        <cfreturn "0">
    </cfif>
</cffunction>
```

- Specify the `returnType="boolean"` attribute in the `cffunction` tag, as in the following example. When a Flash application calls this ColdFusion function as a service, the function returns a valid Boolean True value to Flash.

```
<cffunction name="getBool" access="remote" returnType="boolean">
    <cfset foo = True>
    <cfreturn foo>
</cffunction>
```

Numeric data in ColdFusion

If a ColdFusion page or CFC returns a numeric value without specifically identifying the value as numeric, Flash Remoting MX treats the value as a string when passing it to the responder. For example, if you have the following user-defined function and ActionScript code, Flash displays 22, not 4, in the trace message:

ColdFusion:

```
<cffunction name="getNumber"access="remote">
    <cfreturn 2>
</cffunction>
```

ActionScript:

```
function getNumber_Result ( result )
{
    myVar = (result + 2);
    trace (myVar);
}
```

To prevent such problems, do either of the following:

- Specify the `returnType="numeric"` attribute in the `cffunction` tag, as in the following example:

```
<cffunction name="getNumber" access="remote" returnType="numeric">
    <cfset foo = 2>
    <cfreturn foo>
</cffunction>
```

- Use the `Val` function to explicitly convert the value to a number before you return it, as in the following example:

```
<cffunction name="getNumber" access="remote">
    <cfset foo = Val(2)>
    <cfreturn foo>
</cffunction>
```

If you call either of these `getNumber` functions from Flash, the ActionScript `getNumber_Result` function displays the value 4.

Working with objects

When you pass a Flash object in a service function call, the object's properties are sent to the gateway. In Java environments, an instance of the `flashgateway.io.ASObject` class (which implements `java.util.Map`) represents a Flash object. In .NET environments, an instance of the `FlashGateway.IO.ASObject` class (which implements the `ICollection` interface) represents a Flash object. Therefore, you can pass Flash objects to any service function that accepts a `Map` or `ICollection` argument.

Because Flash Remoting MX transmits data only, the object methods are not available on the server. Similarly, the object properties must be of types that Flash Remoting MX can handle. For example, you cannot include a Flash `RecordSet` object in an object that you pass to a service function, because Flash Remoting MX cannot convert the `RecordSet` object to a data type on the server.

When you return an object from the server to Flash, Flash Remoting MX sends the contents of the object's data properties to Flash as a Flash object. In Flash, you can access any of the object's properties that are of types that can be converted to Flash types.

The following sections cover two special cases of objects: `ActionScript` typed objects and `Serializable` Java objects.

Working with `ActionScript` typed objects

If you use the `Object.RegisterClass` method to register an object in `ActionScript`, you create a typed object. Typed objects are useful in Flash applications for creating subclasses of Flash objects. You can use typed objects in calls to Flash Remoting service functions.

If you use an instance of the object type in a service function call, the `Flashgateway.IO.ASObject` object represents the argument on the server includes the object type name.

For example, the following `ActionScript` creates a typed object and uses it in a service function:

```
//Make a class (Class constructor)
myClass = function()
{
    this.Value1 = "Test1";
}
//Register the class definition
Object.registerClass("testClass", myClass);

//Send instance of registered class to a Flash Remoting gateway service
myService.myFunction(new testClass());
```

When the service function on the application server receives this request, the argument is an object of type `flashgateway.io.ASObject` in Java and `ColdFusion`, or `FlashGateway.IO.ASObject` in .NET environments. The service function can access the class type name, `testClass`, using the object's `getType` method in Java or `ColdFusion` or the `ASType` property in .NET.

When a service function must create a new typed object to return to Flash Remoting MX, it creates an object of type `flashgateway.io.ASObject` in Java or ColdFusion, or of type `FlashGateway.IO.ASObject` in .NET environments. The service function uses the object's constructor or `setType` method in Java, `setType` method in ColdFusion, or the `ASType` property in .NET to set the class type name to the type specified in the `ActionScript Object.registerClass` method.

When the Flash client receives the typed object from the service function, Flash runs the constructor for the type and attaches all the object's prototype functions.

The following example shows a Java class service function that creates a typed object and returns it to Flash:

```
package mycompany.flash;
import flashgateway.io.ASObject;
public class MyFlashService
{
    public MyFlashService()
    {
    }

    public ASObject getFlashObject()
    {
        ASObject aso = new ASObject("MyFlashObject");
        aso.put("first", "apple");
        aso.put("second", "banana");
        return aso;
    }
}
```

Note that this example specifies the object type, `MyFlashObject`, in the constructor.

To create a Flash typed object in ColdFusion, use the `cfobject` tag or the `CreateObject` function, specifying the type as `Java` and class as `flashgateway.io.ASObject`. Then use the object's `setType` method to set the Flash object type name. The following CFML is the equivalent to the Java code:

```
<cffunction access="remote" name="getFlashObject">
    <cfobject type="JAVA" class="flashgateway.io.ASObject" name="myObj"
        " action="CREATE" >
    <cfset myobj.setType("MyFlashObject")>
    <cfset myobj.put("first", "apple")>
    <cfset myobj.put("second", "banana")>
    <cfreturn myobj>
</cffunction>
```

Working with serializable Java objects

If a service function returns an object that implements the Java Serializable interface, its *public* and *private* properties are available as ActionScript properties. For example, a Java service method might return the following JavaBean as the result of a Flash Remoting method invocation. In this case, all three private properties, `text`, `recipient`, and `server`, are available to Flash.

```
public class Message implements java.io.Serializable
{
    private String text;
    private String recipient;
    private String server;

    public Message()
    {
        this.text = "Default message";
        this.recipient = "user@macromedia.com";
        this.server = "smtp.macromedia.com";
    }

    public String getText(){return this.text;}
    public void setText(String t){this.text = t;}

    public String getRecipient(){return this.recipient;}
    public void setRecipient(String r){this.recipient = r;}

    public Message getMessage()
    {
        return this;
    }
}
```

You can use the following ActionScript to set and get the result object's properties. (For brevity, this example omits the code that configures the network connection and service object.)

```
myBeanService.setText("Hello from Me.");
myBeanService.setRecipient("me@macromedia.com");
myBeanService.getMessage();

function getMessage_Result(result)
{
    myMessageText.text = result.text;
    myServerInfo.text = result.server
}
}
```

In this case, ActionScript does not get the value of the `Message.text` property by explicitly calling the `getMessage` method, but directly from the properties returned from the Flash gateway on the server.

Working with RecordSet objects

Using Flash Remoting MX, you can return `RecordSet` objects from application servers, manipulate the records in the `RecordSet` object, and display information from the records in a Flash application. Typically, application servers create record sets from the results of a database query. Some of the uses for `RecordSet` objects in ActionScript include the following:

- Providing product catalog, employee directory, or other information from an application server database query, and browsing the results in a Flash application
- Downloading a set of product options from the application server, and then using the data to build an online catalog in a Flash application
- Retrieving personal data, such as buddy lists or e-mail messages, that are stored in a database, and displaying the lists or e-mail messages in a Flash application

About record sets

A **record set** is a two-dimensional data table. The rows of the table correspond to individual data **records**, such as the data for a particular product or employee. The columns of the table correspond to different **fields** of a record, such as an employee's title or a product color. The following table shows a sample record set structure:

lastName	firstName	emailAddress	telExt
Smith	Dave	dave.tomlin@macromedia.com	3456
Basham	Meredith	meredith.neville@macromedia.com	7890
Card	Sean	sean.carr@macromedia.com	1234
Randolph	Themis	themis.cripps@macromedia.com	5678
Sykes	Andrew	andrew.gruber@macromedia.com	9012

A `RecordSet` **object** represents a record set in Flash. It contains the following elements:

- An array of records
- The names of the columns
- A reference to the application server, if the record set is pageable

Note: For information on pageable record sets, see “Delivering `RecordSet` data to Flash applications in ColdFusion MX” on page 53.

Typically, service functions return `RecordSet` objects to your Flash application. However, you can also use ActionScript `RecordSet` methods to create and manage record sets directly in ActionScript. The ability to create a `RecordSet` object enables you to create custom client-side data structures for use in Flash UI Components. For more information on using `RecordSet` methods, see the following section, “`RecordSet` methods”.

You access record set rows using the row **index**, much like in an array. The first record is at index 0, the second record is at index 1, and so on. Record indexes are relative. If you insert a record into a record set, all the indexes of all records in the `RecordSet` object starting with the index at which you insert the new record get incremented by one.

RecordSet object records also have unique IDs that are never changed. If you insert a record in a RecordSet object, it gets a new unique ID and all other record IDs are unchanged. If you delete a record, its ID is deleted and is not reused. Flash Remoting MX uses this ID internally, and you cannot use it to access a record, but you can use the RecordSet.getItemID method to determine the ID for any record.

Note: You cannot send RecordSet objects to the application server.

RecordSet methods

You can use the following methods to create and manage RecordSet objects

Method	Description
Constructor for RecordSet	Creates a new local RecordSet object.
RecordSet.addItem	Inserts a record into the RecordSet object.
RecordSet.addItemAt	Inserts a record into the RecordSet object at the specified index.
RecordSet.addView	Defines an object that will receive notifications when the RecordSet object changes.
RecordSet.filter	Creates a new RecordSet object that contains selected records from the original RecordSet object.
RecordSet.getColumnNames	Returns the names of all the columns of a RecordSet object.
RecordSet.getItemAt	Returns a record if the index is valid and the record is available.
RecordSet.getItemID	Returns the record ID of a record.
RecordSet.getLength	Returns the number of records in a RecordSet object.
RecordSet.getNumberAvailable	Returns the number of records that have been downloaded from the server.
RecordSet.isFullyPopulated RecordSet.isLocal	Determine whether a RecordSet object is fully available on the client system. (Both methods are equivalent.)
RecordSet.removeAll	Removes all records from the RecordSet object.
RecordSet.removeItemAt	Removes the specified record from the RecordSet object.
RecordSet.replaceItemAt	Replaces a record at the specified index.
RecordSet.setDeliveryMode	Changes the delivery mode of a pageable record set from an application server.
RecordSet.setField	Replaces one field of a record with a new value.
RecordSet.sort	Sorts all the records using a comparison function that you specify as an argument to the method.
RecordSet.sortItemsBy	Sorts all records in the RecordSet object in ascending or descending order, according to the current locale's sorting order.

The following sections describe how you can use these methods to create and manage `RecordSet` objects in ActionScript code.

Using RecordSet methods and properties

The following sections describe how to use `RecordSet` methods to manage `RecordSet` objects.

Creating RecordSet objects

Most `RecordSet` objects are returned by service functions, so you do not typically have to create them. However, you can use the `RecordSet` object constructor to create a record set directly in ActionScript. For example, the following line creates a new `RecordSet` object with two columns: `DepartmentID` and `DepartmentName`. This `RecordSet` object does not contain any data:

```
myRecordSet = new RecordSet(["DepartmentID", "DepartmentName"]);
```

Getting values and information from RecordSet objects

The following sections describe how to get record set values and information.

Getting record set data values

To get a specific record in the `RecordSet` object, specify the record's 0-based index in the `getItemAt` method. For example, the following line gets the third record in a record set:

```
myRecord = myRecordSet.getItemAt(2);
```

To get the value of a specific field in a record, use the column name as a property of the record. For example, use the following line to get the value of the `DepartmentName` field of the fourth record in the `RecordSet` object named `myRecordSet`:

```
myDept = myRecordSet.getItemAt(3).DepartmentName;
```

Note: Because the `RecordSet` object is a subclass of the ActionScript `DataProvider` class, you can also use `RecordSet` objects directly with any object that takes a `DataProvider`, such as the `ComboBox` Flash UI Component. For more information on using `RecordSet` objects with Flash UI components, see “Using Flash MX UI components with `RecordSet` objects” on page 54.

Getting information about a RecordSet object

To get information about a `RecordSet` object, use the following methods:

Method	Description
<code>RecordSet.getColumnNames</code>	Returns an array of the names of the columns of a <code>RecordSet</code> object.
<code>RecordSet.getItemID</code>	Returns the record ID used internally by Flash Remoting MX to identify the record.
<code>RecordSet.getLength</code>	Returns the number of records in a <code>RecordSet</code> object.

Method	Description
<code>RecordSet.getNumberAvailable</code>	Returns the number of records that have been downloaded from the server.
<code>RecordSet.isFullyPopulated</code> <code>RecordSet.isLocal</code>	Returns true if the <code>RecordSet</code> object was created locally using the <code>New</code> operator or if the record set was returned by a Flash Remoting service function and all the data in the record set has been returned from the server. A <code>RecordSet</code> object must be fully populated before you can change its contents or use the <code>RecordSet.filter</code> method. These two methods are currently equivalent.

In the following example, the `theRecordSet` represents a `RecordSet` object:

```
//Get an array of the column names and convert it to a comma delimited list
columns = theRecordSet.getColumnNames().join();
//The total number of records in the RecordSet
recordcount = theRecordSet.getLength();
//The number of records that are currently available to the Flash client
recordsavail = theRecordSet.getNumberAvailable();
//Have all the records been returned? (Only true if recordcount == recordsavail)
allthere = theRecordSet.isFullyPopulated();
```

Changing record set data

After you download all records into the `RecordSet` object or create a new `RecordSet` object in ActionScript, you can use the `RecordSet` ActionScript class data-editing methods to insert, update, and remove records. Changes to the `RecordSet` object in the Flash application are not propagated back to the application server. To insert, update, or remove records in a database, you must call application server methods or pages using service functions.

Adding records to the `RecordSet` object

To add items to a `RecordSet` object, use the `addItem` or `addItemAt` methods. The `addItem` method adds a record at the end of the record set. The `addItemAt` method inserts a record at the specific index location; the indexes of all the other records in the `RecordSet` object are automatically incremented by one. For example, the following adds a single record at the beginning of the `myRecordSet` object:

```
var newRecord = {DepartmentID: "BA1EA7FF0-7D79-32D3-A9280050042189548",
    DepartmentName: "Technical Publications"};
myRecordSet.addItemAt(0, newRecord);
```

Removing records from the `RecordSet` object

To remove records from a `RecordSet` object, use the `removeItemAt` and `removeAll` methods. The `removeItemAt` method removes a record at a specific index location, and the `removeAll` method removes all records from the `RecordSet` object. For example, the following line removes the record at the first index location (0). The indexes of all the other records in the `RecordSet` object are automatically decremented by one:

```
theRecordSet.removeItemAt(0);
```

Replacing and renaming records in the RecordSet object

To replace a record in a `RecordSet` object, you use the `replaceItemAt` method. To replace a specific field in a record in a `RecordSet` object, use the `setField` method. For example, the following code replaces the contents of the third record in the `theRecordSet` `RecordSet` object with the contents of the `newRecord` variable. It then replaces the contents of the third record's `DepartmentName` field:

```
var newRecord = {DepartmentID: "BA1EA720-7D79-11D3-A9280050042189548",
    DepartmentName: "Complaints"};
theRecordSet.replaceItemAt(2, newRecord);
theRecordSet.setField(2,"DepartmentName","Compliments");
```

Using notifications with RecordSet objects

You can notify any `ActionScript` object of changes in a `RecordSet` object's internal state. For example, if you associate a `RecordSet` object with a `ListBox` component, and the record set gets sorted into a different order, Flash can notify the `ListBox` component that the record order changed, so the `ListBox` component can redraw itself according to the new order.

Similarly, a `RecordSet` object can notify its associated `ListBox` component when all records arrive from the server. Then, if required, the `ListBox` can redraw itself to update its display.

Note: Flash UI components such as `ListBox` incorporate notifications as a standard part of their use of the `RecordSet` `ActionScript` class. The `addView` method is only necessary if you need to receive notifications in your own `ActionScript` code, for example, if you create your own UI component.

To set up a notification event for an `ActionScript` object, use the `RecordSet` `addView` method and specify the object to notify when the `RecordSet` object changes. You can specify any object that receives change notifications in the `addView` method, as in the following example:

```
myRecordSet.addView(contact_grid);
```

In this example, the `myRecordSet` object represents a record set returned from Flash Remoting MX, and the `addView` method tells Flash Remoting MX to notify the `contact_grid` object whenever the `theRecordSet` object changes.

The object that receives the notification must include a `modelChanged` callback function to handle the notification. Whenever the `RecordSet` object changes, the `modelChanged` function gets called with a message object that consists of one or more entries, as follows:

- The first, `event`, entry identifies the type of event or change that was made to the record.
- For some types of events, the message object also includes entries identifying the first and last record in the record set that were affected by the event.

The following table describes the event messages:

Message object	Description
{event: "sort"}	The RecordSet object has been sorted.
{event:"updateAll"}	The RecordSet object has changed in some way, such as a new view being added.
{event:"addRows", firstRow:xxx, lastRow:yyy}	Row numbers xxx through yyy have been added.
{event:"updateRows", firstRow:xxx, lastRow:yyy}	Row numbers xxx through yyy have changed in some way.
{event:"allRows"}	All records have arrived from the server.
{event:"fetchrows", firstRow:xxx, lastRow:yyy}	Row numbers xxx through yyy have been requested from the server, but have not arrived yet.

The following example creates a `modelChanged` function that displays a trace message with the event type and sets it up as the callback handler for changes to the `myRecordSet` object:

```
function modelChanged(info)
{
    trace("Caught modelChanged event: " + info.event);
}

myRecordSet = new RecordSet(["DepartmentID", "DepartmentName"]);
myRecordSet.AddView(this);
```

Sorting and filtering record sets

The `RecordSet` class has methods for sorting an existing `RecordSet` object or creating a new `RecordSet` object from an existing one by applying a selection (filter) function.

Sorting record sets

To sort the records of a `RecordSet` object, use the `sort` and `sortItemBy` methods. The `sortItemBy` method performs an ascending (the default) or descending sort on the records in the `RecordSet` object. The `sort` method requires a comparison function as an argument, and uses that function to sort the records. The `sort` method can be substantially slower than the `sortItemBy` method.

Sorting a `RecordSet` object changes the order of the records in the object, but does not otherwise change the object. Subsequent `getItemAt` method calls return records according to the new order. After you sort a `RecordSet` object that was returned from an application, the object no longer reflects the order of the records on the server-side record set.

The following one-line example sorts the `myRecordSet` object according to the value of the `DepartmentName` field in descending order:

```
myRecordSet.SortItemsBy("DepartmentName", "DESC")
```

The following example shows how you can create a function that sorts the contents of a `ListBox` UI component:

```
function SortBy(sorter)
{
    temp = sorter.getSelectedItemAt();
    sort_this = temp.data;
    theRecordSet.sortItemsBy(sort_this);
}
```

In this example, `sorter` represents a `ListBox` UI component in the Flash application. In the `sortBy` function, the `getSelectedItem` method returns the item selected in the `ListBox`. Next, the function assigns the selected item to the `sort_this` variable. Finally, the `sortItemsBy` method sorts the records according to the contents of the `sort_this` variable. For an example of using the `sort` method, see “`RecordSet.sort`” on page 184.

Note: For `RecordSet` objects containing 2,000 or fewer records, the `sort` method generally takes less than one second to finish on a Pentium 3 computer. The length of time to sort `RecordSet` objects increases rapidly as the number of records grows.

Filtering an existing `RecordSet` object to create a new `RecordSet` object

The `filter` method creates a filtered view of a `RecordSet` object that contains only records that conform to a set of rules specified by a selection (filter) function that you define. Unlike the `sort` method, the `filter` method creates a new `RecordSet` object. The original `RecordSet` object and its records remain unchanged.

The selection function that you define takes a record as the first argument, and can optionally take a second argument to use to determine how to select the records. The function must return a Boolean `true` or `false` value. Flash Remoting MX includes records for which the selection function returns `true` in the filtered `RecordSet` object. When you call the `filter` method, you pass it your selection function and, optionally, the value to use as the selection function’s second argument.

The following example filters a `RecordSet` object to produce a new `RecordSet` object with records that have contact fields that start with a specific letter:

```
var mySelectionFunction = function(aRecord, letter)
{
    return (aRecord.contact.charAt(0) == letter);
}
contact_grid.setDataProvider(myRecordSet.filter(mySelectionFunction, theLetter));
```

In this example:

- The `contact_grid` variable represents a `ListBox` UI component.
- The `myRecordSet` variable represents a record set that has been retrieved by calling a service function.
- The `theLetter` variable represents user input in the Flash application (a single letter).
- The `mySelectionFunction` function takes a record and a letter and returns `true` if the first letter of the record matches the specified letter.
- The `myRecordSet.filter` method filters the `myRecordSet` object using the `mySelectionFunction` function to create a new `RecordSet` object. Only those records for which the `mySelectionFunction` function returns `true` are included in the new `RecordSet` object.

- The `Contact_grid` component's `setDataProvider` method uses a filtered copy of `myRecordSet` object generated by the `myRecordSet.filter` method as the data provider for the `contact_grid` `ListBox`.

Note: For `RecordSet` objects that with 2,000 or fewer records, the `filter` method generally takes less than one second to finish on a Pentium 3 computer. The length of time to filter `RecordSet` objects increases rapidly as the number of records grows.

Delivering RecordSet data to Flash applications in ColdFusion MX

By default, Flash Remoting MX returns a `RecordSet` object to the Flash client in a single response when the application server finishes retrieving the data.

In ColdFusion MX, if you expect to return a large record set and the available data transmission speeds are slow, such as when using dial-up modem, you can choose to return a record set from the application server in increments. Incremental record sets are also known as **pageable record sets**.

When you use pageable record sets, the following events occur:

- On the server, the record set is held in session data, and server-side Flash Remoting `RecordSet` service provides access to the records.
- On the client, the `RecordSet` object can download records when needed by the Flash application.

Flash Remoting MX can deliver pageable record set data to your application in three modes, as described in the following table:

Data delivery modes	Description
ondemand (default)	When you access a particular record using the <code>getItemAt</code> method, the <code>RecordSet</code> object requests the record from the server-side Flash Remoting <code>RecordSet</code> service.
page	When you use the <code>getItemAt</code> method to access a record, the <code>RecordSet</code> object fetches data one or more pages at a time. You specify the number of records per page when you set the delivery mode. The default page size is 25 records. You can also tell Flash Remoting MX to get, or prefetch , a number of pages of data that follow the page that contains the requested data. Flash Remoting MX will store these additional pages in the client if it has not already fetched them in a previous request. The default prefetch value is 0 (only get the page with the requested data). For example, if you specify a page size of 15 records and a number of pages to prefetch of 3, Flash Remoting MX automatically fetches 45 records when you make your first request for a record in the data set. If you then request a record on the second page that was returned, Flash Remoting MX prefetches an additional 15 records.
fetchall	As a background activity, the <code>RecordSet</code> object fetches the entire contents of the record set from the server, starting when the service function that retrieves the record set on the application server returns. You can specify a page size when you set the delivery mode to <code>fetchall</code> . Flash Remoting continues requesting one page at a time from the server until all pages have been returned. The default page size is 25 records.

You change a `RecordSet` object's delivery mode by calling the `setDeliveryMode` method, as in the following example:

```
if (config_panel.deliveryMode.getData() == "page")
{
    theRecordSet.setDeliveryMode("page", contact_grid.getRowCount(), 1);
}
else if (config_panel.deliveryMode.getData() == "fetchall")
{
    theRecordSet.setDeliveryMode("fetchall", 10);
}
else
{
    theRecordSet.setDeliveryMode("ondemand");
}
```

In this example, the `RecordSet` object represents the record set returned from Flash Remoting MX, and the `config_panel` object represents a Flash movie clip. Using the `deliveryMode.getData` method, the code evaluates the delivery mode specified by `config_panel`, and uses the `setDeliveryMode` method to set the mode of delivery for the `RecordSet` object.

If the `config_panel` object specifies `page` mode, the `setDeliveryMode` method tells Flash Remoting MX to set the page size to the number of rows in the `contact_grid` movie clip and to prefetch one additional page beyond the page with the requested data, if it is not in memory.

If the `config_panel` object specifies `fetchall` mode, the `setDeliveryMode` method tells Flash Remoting MX to set the page size to 10 records and start getting the records, one page at a time (if they are not already on the client), until the entire record set has been fetched.

For all delivery modes, after a record is received from the application server, it is held inside the `RecordSet` object. Any subsequent `getItemAt` calls immediately return the record. Any `getItemAt` calls for records that the client has not yet received fetch the requested record as soon as possible and return a `fetch pending` message.

Using Flash MX UI components with RecordSet objects

Many Flash MX UI components can use `RecordSet` objects to provide both label and data information. These components use Flash `DataProvider` objects to supply the following information:

- Label values that appear to the user
- Corresponding data values that are available by using the component's `getValue` method after the user selects a label

Objects that can use `DataProvider` objects are often referred to as **data consumers**. Flash components that can be data consumers currently include the following:

Standard Flash UI components

<code>FListBox</code>	<code>FComboBox</code>
-----------------------	------------------------

Flash UI Components Set 2

<code>FTicker</code>	<code>FTreeNode</code>
<code>FTree</code>	

Charting Components

<code>FBarChart</code>	<code>FPieChart</code>
<code>FLineChart</code>	

The Flash UI Components Set 2 and Charting components are downloadable from the Flash exchange at <http://dynamic.macromedia.com/bin/MM/exchange/main.jsp?product=flash>.) Additional UI objects might be available at the Macromedia Flash Exchange.

The following sections describe how to use the `RecordSet` objects with these components.

Using `RecordSet` objects directly

You can use `RecordSet` objects directly in the `setDataProvider` method of a data consumer component to specify that the `RecordSet` object provides the component's values.

The following example is a result handler for a `getProductList` service function that gets a single-column record set that contains product names. It populates a `ListBox` component with the returned `RecordSet` object's records:

```
function getProductList_Result (result)
{
    catalogListBox.setDataProvider(result);
}
catalogService.getProductList();
```

By default, each label value is a comma-delimited string that consists of the contents of one record's fields; the data values do not get set. However, you can use a `RecordSet` object directly in the `setDataProvider` method to provide both the list and data values if the record set has two columns and the column names are `label` and `data`. For example, the following SQL code produces a record set that, when passed to the preceding `getProductList_Result` function, populates the `catalogListBox` object with both label and data values:

```
SELECT COST_CENTER AS DATA, DESCRIPTION AS LABEL
FROM EMPDB_DEPARTMENT
WHERE STATUS='Active'
```

Using DataGlue methods

The DataGlue **methods**, `bindFormatStrings` and `bindFormatFunction`, let you specify how a `RecordSet` object supplies the contents of both the data and value fields of a data consumer. The DataGlue methods provide substantial flexibility in formatting the contents of the labels and data, as follows:

- The `bindFormatStrings` method lets you independently specify strings that contain any number of record set fields and other data as the sources of the label and data contents.
- The `bindFormatFunction` method lets you specify any function to provide the data to the data consumer. The function must take a record as an argument and return an object that consists of two entries: a label field and a data field. The function has full flexibility in using the record contents to generate and format the label and data values.

The DataGlue methods do not make a copy of the `DataProvider` object. However, the data is fetched from the data provider as needed by the component.

Using the `bindFormatStrings` method

The `bindFormatStrings` method lets you freely format record fields and string data. For example, the following method uses values from two fields to generate the labels and from three fields to generate the data values:

```
DataGlue.bindFormatStrings (myComboBox, myRecordSet, "#parkname# (#parktype#)",
    "#city#, #state# #zipcode#");
```

In this example, `myComboBox` represents a `ComboBox` component in the Flash application, and `myRecordSet` represents the `RecordSet` object. The `parkname`, `parktype`, `city`, `state`, and `zipcode` variables represent record field names. The Flash application displays the `parkname` and `parktype` variables in the `ComboBox`. The `city`, `state`, and `zipcode` variables are returned when the user selects the record and ActionScript code uses a `getValue`, `getSelectedItem`, or similar, method.

Using the `bindFormatFunction` method

The `bindFormatFunction` method lets you call a function to format the data for your data consumer. Your function must take a record as its argument and return an object with the following format:

```
{label: labelValue, data: dataValue};
```

For example, the following formatting function takes a record that includes a `parkname` field. It converts the `parkname` text to all lowercase as the label, and uses the field's length as the data. The `bindFormatFunction` method uses the output of this function to populate a Flash UI component:

```
function myFormatFunction ( record )
{
    // the label is the parkname record field, translated to lowercase
    var theLabel = record.parkname.toLowerCase();

    // the data is the length of the parkname record field
    var theData = record.parkname.length;
```

```

    // return the label and value to the caller
    return {label: theLabel, data: theData};
}
//call the bindFormatFunction method
DataGlue.bindFormatFunction(dataView2, result, myFormatFunction);

```

Working with XML

When you use Flash Remoting MX you can use either of the following patterns for handling XML data:

- You do not use XML in Flash. The Flash application sends information to the server using simpler data types, including objects if needed. The service functions can generate and manipulate XML data as necessary. They convert any XML to simpler data types to return to your Flash application.
- You use XML directly in Flash. The service functions get XML from, and return XML to, the Flash application. Your Flash application uses ActionScript XML objects and methods to generate and manipulate XML as needed.

Your decision as to which method to use should depend on whether it is preferable in your environment to do more processing in the server or in the client Flash application. For example, many application servers, such as ColdFusion, provide tools that are specifically optimized for XML manipulation. Therefore, although Flash includes support for XML objects and provides methods for manipulating them, you might find it more efficient to process complex XML on your server and send the processed data to Flash in custom objects, rather than having your service functions return XML to Flash. However, if you are using Flash Remoting MX to call a service that returns XML, you can use the Flash XML methods to access the XML directly.

If you do use XML objects in your Flash application, Flash Remoting MX converts between the Flash XML objects used on the client and the standard XML document object type for the application server: `System.Xml.XmlDocument` in .NET environments, `org.w3c.dom.Document` in Java, and XML document objects in ColdFusion.

The following example sends the contents of two input boxes in an XML object. The first XML element has the contents of the first text input; this element has a single child that has the contents of the second text box. The server echoes the XML back to the Flash application, and the `testDocument_result` result handler concatenates the contents of the two nodes to create a single string for the output box.

```

//Function that runs when the user clicks a "Run XML" button after entering text in
//two text boxes
function testDocument()
{
    //Create the XML document.
    xmlDocument = new XML();
    firstElement = xmlDocument.createElement("TEST");
    firstElement.attributes.message = input1.text;
    secondElement = xmlDocument.createElement("INSIDETEST");
    secondElement.attributes.message = input2.text;
    firstElement.appendChild(secondElement);
}

```

```
xmlDocument.appendChild(firstElement);

//Call the service function and pass it the XML document
flashtestService.testDocument(xmlDocument);
}

//Result Handler callback function to handle the results returned by Flash Remoting
function testDocument_Result(result)
{
    // result is an XML object
    // for this example, get an attribute from the first node
    output.text = result.firstChild.attributes["message"] +
        result.firstChild.firstChild.attributes["message"];
}
}
```

Note: Because the Flash XML object is a standard Flash object and not part of Flash Remoting MX, this document does not cover the details of using XML objects in Flash. For more information on using the Flash XML objects, see the Flash online Help.

CHAPTER 4

Using the NetConnection Debugger

This chapter describes how to use the NetConnection Debugger and the NetDebug and NetDebugConfig classes to debug your Flash Remoting application. It includes a brief description of the NetConnection Debugger interface and descriptions of the types of events that the debugger can display. It also documents how you can use the NetDebug class to control the information displayed in the NetConnection Debugger, including displaying Trace events.

Contents

- About the NetConnection Debugger 60
- NetConnection events 61
- Using the NetConnection Debugger in ActionScript 66

About the NetConnection Debugger

To debug Flash Remoting applications, you use the NetConnection Debugger in the Flash MX authoring environment. The NetConnection Debugger shows calls and responses from the following:

- Flash Player
- Flash Remoting MX
- Flash Communication Server
- Application server

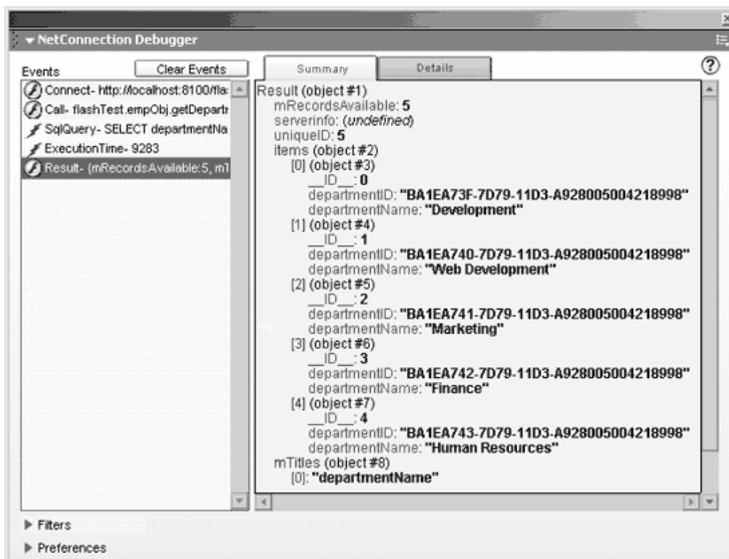
Note: This document does not cover using the NetConnection Debugger with Macromedia Flash Communication Server applications.

To use the NetConnection Debugger with your Flash Remoting application, include the following line in the ActionScript for the first frame of your main movie:

```
#include NetDebug.as
```

To open the NetConnection Debugger, select **Window > NetConnection Debugger** in the Flash MX authoring environment.

The following figure shows the NetConnection Debugger:



This document does not describe the NetConnection Debugger interface in detail. For more information about the NetConnection Debugger interface, see the online Help. To access the online Help, click the blue question mark (?) button in the upper right of the panel.

NetConnection events

The NetConnection Debugger can display information about a variety of events, and you can select which types to display. The following sections describe the event types and the information that the event messages for different types provide.

NetConnection event types

The NetConnection Debugger displays information about events belonging to three categories of event: `client`, `app_server`, and `flashcomm_server`, and each category has one or more specific event types. You can enable or disable all events in a category, or specific event types.

Note: The Network Connection Debugger Filters panel provides the best method to select which events to display. However, you can use ActionScript to enable or disable specific event types for individual NetConnection objects. For more information on using ActionScript to enable or disable event information, see “Configuring debugger output in ActionScript” on page 66.

The following table lists types of events that the NetConnection Debugger displays:

Event type	Description
<code>client</code> (default = true)	Activity information sent from the client. For detailed information on client debug events, see “client event messages” on page 63. You can selectively enable and disable the following subcategories of client events.
<code>trace</code> (default = true)	Client NetConnection trace events. If client trace reporting is enabled, the NetConnection Debugger displays the output of <code>NetConnection.trace</code> and <code>NetDebug.trace</code> methods.
<code>http</code> (default = true)	Client NetConnection events that communicate with the remote source using HTTP. These events include connecting to the server, and Flash Remoting service calls and result returns.
<code>recordset</code> (default = true)	Events associated with the delivery of pageable record set data to the client. These events occur only if the record set is pageable.
<code>rtmp</code> (default = true)	Client NetConnection events that communicate with the remote server using Real-Time Messaging Protocol (RTMP). Communication with the Flash Communication Server uses RTMP. Flash Remoting MX does not use RTMP.
<code>app_server</code> (default = true)	Information about events on the application server. This category includes activities reported by the server itself and Flash Remoting gateway activities. For detailed information on server debug events, see “app_server event messages” on page 64. You can selectively enable and disable the following subcategories of app_server events.
<code>trace</code> (default = true)	Server trace events. Not currently used.

Event type	Description
error (default = true)	Server error events. These include any errors generated during the execution of the Flash Remoting gateway or adapters.
httpheaders (default = false)	Server HTTP header events. When enabled, the NetConnection Debugger reports a server HTTP header event for each HTTP request.
amfheaders (default = false)	Action message Format (AMF) header events. When enabled, the NetConnection Debugger reports a server AMF request header event and a server AMF response header event for each AMF request and response. A single AMF request can include multiple service function calls.
amf (default = false)	Server AMF events. When enabled, the NetConnection Debugger reports a server AMF method call event and a server AMF method response event for each service function call and response.
recordset (default = true)	Events associated with the partial delivery of pageable record set data from the server. These events occur only if the record set is pageable.
coldfusion (default = true)	ColdFusion debug events. When enabled, any debug events enabled in ColdFusion are reported to the NetConnection Debugger. If ColdFusion debugging information is not enabled in ColdFusion, a single debug error event that notifies the developer that ColdFusion debugging information has not been enabled on the server is returned per transaction. For more information on configuring debugging information in ColdFusion, see the ColdFusion documentation.
flashcomm_server/ realtime_server (default = true)	Information about Macromedia Flash Communication Server MX events. This option is not used for debugging Flash Remoting applications. <i>Note:</i> The user interface uses the term <code>flashcomm_server</code> . In ActionScript, you must use <code>realtime_server</code> . You can selectively enable and disable the following subcategory of events.
trace (default = true)	Flash Communication Server trace events.

Common event information

The following table describes the information that the NetConnection Debugger displays for all events. Two fields, `DebugID` and `Protocol`, are not used for trace events initiated by the `Netdebug.trace` method.

Field	Description
DebugID	The <code>DebugID</code> specified in a <code>NetConnection.SetDebugID</code> method for the connection being used. If you do not use the <code>SetDebugID</code> method, Flash Remoting MX sets this ID to a numeric value. Not displayed for <code>NetDebug.trace</code> events.
EventType	A string identifying the type of event being reported.

Field	Description
MovieUrl	The URL of the Flash application associated with the event.
Protocol	The protocol used for communication with the server. The value is http for all Flash Remoting applications. Not displayed for NetDebug.trace events.
Source	The source of the event; always Client.
Time	A numeric millisecond timestamp that indicates when the event occurred.
Date	The date and time when the event occurred, including information about the time value's offset from Universal Time (also known as GMT or UTC).

client event messages

The following table describes the specific client events that the NetConnection Debugger reports, and the information it displays for each event:

Event type	Description	Information fields
Trace	Flash executed a NetDebug.trace or NetConnection.trace method.	Trace The object passed to the NetDebug.trace or NetConnection.trace method. Typically a message.
Connect	The Flash application requested a connection to the gateway.	ConnectString The connection string sent to the gateway. The gateway URL.
AddHeader	The Flash Remoting client added an additional header that is sent with each message to the gateway. This event occurs when you use the NetConnection.SetCredentials method.	HeaderName The type of header being sent, such as Credentials. MustUnderstand A Boolean value that specifies whether the server must understand and process this header before can handle any following headers or messages. HeaderObject The object that will be sent in the added header. For example, the password and user ID.
Result	Flash Remoting MX received a result message from the gateway.	Result The object sent by the gateway in response to the request.
Status	Flash Remoting MX received an error message from the gateway.	Status An error object that describes the error. For more information about the error object, see "The error object" on page 31,

Event type	Description	Information fields
Call	Flash Remoting MX sent a service function call to the gateway.	<p>MethodName The service function being called.</p> <p>Parameters The arguments to the service function call.</p>
Close	Flash Remoting MX closed the connection to the gateway. Initiated by a <code>Netconnection.close</code> method.	None.

app_server event messages

The following table describes the specific `app_server` events that the NetConnection Debugger reports and the information it displays for each event:

Event type	Description	Information fields
Error	An unhandled error occurred in the Flash Remoting gateway or adapter. These errors should not occur in normal processing.	<p>Data Any object associated with the error event. Normally an Exception object.</p> <p>Message A message that describes the error.</p>
HttpRequest Header	The gateway received an HTTP request.	HTTPHeaders The contents of the HTTP request headers.
AmfRequest Header	The gateway received an AMF request message. Each HTTP request received by the gateway includes an AMF request.	AmfHeaders – The contents of the AMF headers. The headers include the Credentials header, if the Flash Application uses the <code>SetCredentials</code> method, and the <code>amf_server_debug</code> header, which contains the NetConnection Debugger <code>app_server</code> debugging settings.
AmfResponse Header	The gateway returned a response to the Flash Remoting client. This event occurs for each HTTP request.	AmfHeaders The contents of the AMF headers sent in the response, if any. The gateway returns a header if the client does not support cookies and URL rewriting is being used for session management.
AmfMethod Call	The gateway received a service function call. A single AMF request can contain multiple AMF method calls.	<p>MethodName The service function being called.</p> <p>Response URI The client-side identifier for the responder. This value is internally generated by Flash Remoting MX.</p> <p>Parameters The arguments passed to the service function.</p>

Event type	Description	Information fields
AmfResponse Call	The Flash gateway sent result data to the client.	<p>MethodName The Response URI identified by the AmfMethodCall message, followed by /onResult.</p> <p>Response URI Normally “(undefined)”.</p> <p>Parameters – The data returned by the service function.</p>
AmfStatusCall	The flash gateway or application server threw an exception.	<p>MethodName The Response URI identified by the AmfMethodCall message, followed by /onStatus.</p> <p>Response URI Normally “(undefined)”.</p> <p>Parameters An error object that describes the error. For more information about the error object, see “The error object” on page 31.</p>
Information	Server or AMF status information.	<p>Message A message that provides information about the status.</p>

ColdFusion event messages

If you enable debugging output in ColdFusion, the NetConnection Debugger can report a subset of the debugging information that is normally displayed by the ColdFusion server. The NetConnection Debugger can report the following ColdFusion debugging information:

- Template (for CFM pages only, not CFCs)
- Execution time
- SQL query
- cftrace tag output
- Exceptions (for exceptions caught in ColdFusion only)
- HTTP (for cfhttp tag)

Note: The NetConnection Debugger displays an information event reporting that ColdFusion debugging is not enabled on the server if debugging is enabled but no debugging information is available, for example, if an error occurs when running a CFML service function.

For more information on ColdFusion debugging information, see *Developing ColdFusion MX Applications with CFML*.

Flash Communication Server events

If you enable Flash Communication Server (flashcom_server) debugging events, the NetConnection Debugger displays an event message when the Flash Communication Server application calls a trace function. The message includes the trace call parameter and the Flash Communication Server logging API Info object for the event.

Using the NetConnection Debugger in ActionScript

Flash Remoting MX provides several objects and methods that let you control the information that appears in the NetConnection Debugger. These tools include the following:

- The `NetDebug.trace` method, to display trace messages in the debugger
- `NetConnection` object methods, to provide debugging information for specific connections
- The `NetDebugConfig` object, to select the information that the debugger displays

Using the `NetDebug.trace` method

The `NetDebug.trace` method displays a trace message in the NetConnection Debugger. For example, the following trace method displays a trace message with the text “I just created `myService`.” in the NetConnection Debugger:

```
NetDebug.trace("I just created myService.");
```

The argument to the trace method is not limited to a string. It can be a Flash Object. For example, you can use the trace method to report the values of multiple variables, as in the following code:

```
NetDebug.trace({arg1value:arg1, arg2value:arg2});
```

Using connection-specific debugging methods

If your application uses multiple connections, you can use the debugging-related methods of each `NetConnection` object to debug the individual connections.

The `setDebugID` method creates an ID that is displayed in the NetConnection Debugger output for events associated with the specific `NetConnection` object; for example:

```
gatewayConnection.setDebugID("Gateway Connection");
```

You can send a trace message that includes the connection ID by using the `GatewayConnection` object’s trace method, as in the following example:

```
myGatewayConnection.trace("I just created myService2 over this connection.");
```

Additionally, the `getDebugID` method returns the ID set by the `setDebugID` method, and the `getDebugConfig` method returns the `NetDebugConfig` object for the specific connection. For more information on using the `getDebugConfig` method, see the following section, “Configuring debugger output in ActionScript”.

Configuring debugger output in ActionScript

You can specify the information that the NetConnection Debugger displays by selecting options on the debugger Filters panel, or you can do it programmatically in ActionScript. This document describes how to use ActionScript. For information on using the Filters panel, see the Flash online Help.

In ActionScript, you configure NetConnection Debugger output for each connection individually. As a result, if your Flash Remoting application has multiple connections, you can configure different levels of debugging detail for each connection.

To configure debugging output:

- 1 Use the `NetConnection` `getDebugConfig` method to get the connection's `NetDebugConfig` object.
- 2 Set the required property of the `NetDebugConfig` object. The table in “`NetConnection` event types” on page 61 specifies the properties you can set.

You can combine both steps in a single line, as in the following example, which turns off application server AMF debugging messages:

```
gatewayConnection.getDebugConfig().app_server.amf = false;
```

Note: You can only use this technique to select or disable individual event types. You cannot use a single call to select or disable all debugging information for the client, application server, or Flash Communication Server.

CHAPTER 5

Using Flash Remoting MX with ColdFusion MX

To use Macromedia Flash Remoting MX with Macromedia ColdFusion MX, you build ColdFusion pages or ColdFusion components. In ColdFusion pages, you use the Flash variable scope to interact with Flash applications. ColdFusion components natively support Flash interaction. In addition, you can use ColdFusion's server-side ActionScript functionality, which lets you query databases and perform HTTP operations in ActionScript files on the server.

Contents

- Using Flash Remoting MX with ColdFusion pages..... 70
- Using Flash Remoting MX with ColdFusion components 77
- Using Flash Remoting MX with server-side ActionScript..... 82
- Calling web services from Flash Remoting MX..... 86
- Securing access to ColdFusion from Flash Remoting MX..... 88
- Handling errors with ColdFusion 90

Using Flash Remoting MX with ColdFusion pages

When developing ColdFusion pages for use with Flash Remoting MX, you need to know how to do the following:

- Determine the Flash service name for the ColdFusion page
- Pass parameters back and forth between the Flash application and the ColdFusion page
- Return information to a Flash application from a ColdFusion page

The following sections describe how to perform these actions.

Determining the Flash service name

When building a ColdFusion page called from Flash applications, the directory name on the server containing the ColdFusion page translates to the service name called in ActionScript from Flash. The ColdFusion page names contained in that directory translate to service functions in ActionScript.

For example, you create a ColdFusion page named `helloWorld.cfm` in the directory `helloExamples` under your web root (*web_root*/`helloExamples`). You then use the following ActionScript in your Flash application to call `helloWorld.cfm`:

```
#include "NetServices.as"
NetServices.setDefaultGatewayUrl("http://localhost/flashservices/gateway");
gatewayConnection = NetServices.createGatewayConnection();
CFMService = gatewayConnection.getService("helloExamples", this);
CFMService.helloWorld();
```

To specify subdirectories of the web root directory or a virtual directory, use package dot notation. If `helloWorld.cfm` was in the directory *web_root*/`helloExamples/ColdFusion`, you use the following ActionScript to create the service:

```
CFMService = gatewayConnection.getService("helloExamples.ColdFusion", this);
```

Remember to use periods to delimit directory names for `getService()`.

Using the Flash scope to pass parameters to ColdFusion pages

ColdFusion MX defines a scope called **Flash** that you use to access parameters passed from Flash applications and return values to Flash applications.

The Flash scope has several predefined variables that you can use to pass information, as described in the following table:

Variable	Description	For more information
Flash.Params	Array containing the parameters passed from the Flash application to the ColdFusion page. If you do not pass any parameters, Flash.Params still exists, but it is empty.	See “Using Flash.Params to access parameters in a ColdFusion page” on page 71.
Flash.Result*	Result returned to the Flash application from the ColdFusion page.	See “Returning results to ActionScript” on page 74.
Flash.Pagesize	Number of records in each increment of a record set returned to Flash from a ColdFusion page.	See “Returning record sets to Flash” on page 75.

* Due to ActionScript's automatic type conversion, do not return a boolean literal to Flash from ColdFusion. Return 1 to indicate `true`, and return 0 to indicate `false`.

When you call a ColdFusion page from a Flash application, the Flash Remoting gateway converts ActionScript data types to ColdFusion data types. The data type of any results returned from ColdFusion to the Flash application are converted to ActionScript data types. For more information on this conversion, see Chapter 3, “Using Flash Remoting Data in ActionScript” on page 35.

Using Flash.Params to access parameters in a ColdFusion page

The `Flash.Params` array contains one element for each parameter passed from ActionScript, in the order that the parameters were passed to the ColdFusion page. You use standard ColdFusion array syntax to reference the parameters.

For example, the following ActionScript call passes three parameters:

```
myService.myMethod(param1, param2, param3);
```

In your ColdFusion page, you access these parameters using `Flash.Params`, as follows:

```
<cfset p1=Flash.Params[1]>
<cfset p2=Flash.Params[2]>
<cfset p3=Flash.Params[3]>
```

The following ActionScript calls a ColdFusion page to execute a query. The ActionScript passes a single parameter to the ColdFusion page:

```
NetServices.setDefaultGatewayUrl("http://localhost/flashservices/gateway");
gatewayConnection = NetServices.createGatewayConnection();
myService = gatewayConnection.getService("doc_code", this);

myService.query1param("RipperStik");
```

In your ColdFusion page, you access the parameter using `Flash.Params`, as shown in the following example:

```
<cfquery name="flashQuery" datasource="exampleapps" >
    SELECT ItemName, ItemDescription, ItemCost
    FROM tblItems
    WHERE ItemName='#Flash.Params[1]#'
</cfquery>

<cfset Flash.Result=flashQuery>
```

Because ColdFusion converts an ActionScript data type to the corresponding ColdFusion data type, you can perform CFML type-specific operations on the parameter. Therefore, if a parameter passed from a Flash application is an ordered array, you can perform all CFML array operations on the parameter.

For example, if you pass an array using the following ActionScript:

```
var array1 = new Array();
array1[0] = "zero";
array1[1] = "one";
myService.myMethod(array1, param2, param3);
```

You access the elements in the array in your ColdFusion page using ColdFusion array notation, as follows:

```
<cfset arrayElement1=Flash.Params[1][1]>
<cfset arrayElement2=Flash.Params[1][2]>
```

Note: While ActionScript starts the array index at zero, ColdFusion array indexes start at one.

ActionScript also supports named, or associative, arrays. These arrays have the following form:

```
var struct1 = new Array();
struct1["zero"] = "banana";
struct1["one"] = "orange";
myService.myMethod(struct1, param2, param3);
```

ColdFusion converts associative arrays into ColdFusion structures. To access the associative array elements from `Flash.Params`, you use structure notation, as follows:

```
<cfset structElement1=Flash.Params[1].zero>
<cfset structElement2=Flash.Params[1].one>
```

Note: You can use most of the CFML array and structure functions on ActionScript collections. However, one CFML function, `StructCopy`, does not work with ActionScript collections.

The following table describes ActionScript collections and how to access them in ColdFusion pages:

Collection	ActionScript example	Notes
Strict array	<pre>var myArray = new Array(); myArray[0] = "zero"; myArray[1] = "one"; myService.myMethod(myArray);</pre>	<p>The Flash Remoting service converts the array argument to a ColdFusion MX array. All CFML array operations work as expected.</p> <pre><cfset p1=Flash.Params[1][1]> <cfset p2=Flash.Params[1][2]></pre>
Named or associative array	<pre>var myStruct = new Array(); myStruct["zero"] = "banana"; myStruct["one"] = "orange"; myService.myMethod(myStruct);</pre>	<p>In ActionScript, named array keys are not case-sensitive.</p> <pre><cfset p1=Flash.Params[1].zero> <cfset p2=Flash.Params[1].one></pre>
Named arguments using object initializer	<pre>myService.myMethod({x:1, y:2, z:3});</pre>	<p>Provides a convenient way of passing named arguments to ColdFusion pages. Access these arguments in ColdFusion pages as members of the Flash scope.</p> <pre><cfset p1=Flash.x> <cfset p2=Flash.y> <cfset p3=Flash.z></pre>

Accessing ActionScript objects

ActionScript supports the object initializer syntax when calling a function. For example, the following function call passes two parameters as objects:

```
myService.myMethod({x:1, y:2});
```

In this example, the function passes *x* with a value of 1 and *y* with a value of 2.

In your ColdFusion page, you can access objects using the object name, as in the following example:

```
<cfset param1=Flash.x>
<cfset param2=Flash.y>
```

You can also pass arrays and structures using this syntax, as follows:

```
var array1 = new Array();
array1[0] = "zero";
array1[1] = "one";
```

```
var struct1 = new Array();
struct1["zero"] = "banana";
struct1["one"] = "orange";
```

```
myService.myMethod({x:array1, y:struct1});
```

You access *x* and *y* in your ColdFusion page using ColdFusion array and structure notation, as follows:

```
<cfset arrayElement1=Flash.x[1]>
<cfset arrayElement2=Flash.x[2]>
<cfset structElement1=Flash.y.zero>
<cfset structElement2=Flash.y.one>
```

You can pass ActionScript objects to ColdFusion pages. The following ActionScript defines an object:

```
var myObj = new Object();
myObj.x = "one";
myService.myMethod(myObj);
```

In ColdFusion, you access the object elements using named parameters in the Flash scope, as follows:

```
<cfset p1=Flash.myObj>
```

Returning results to ActionScript

In ColdFusion pages, only the value of the `Flash.Result` variable is returned to the Flash application. While you are limited to returning a single variable to the Flash application, that variable can contain a single value, an array, a structure, or a record set returned from a ColdFusion query.

For more information about converting data types between ColdFusion and Flash, see Chapter 3, “Using Flash Remoting Data in ActionScript” on page 35.

The following procedure creates the service function `helloWorld`, which returns a structure that contains simple messages to the Flash application.

To create a ColdFusion page that returns a structure:

- 1 Create a folder in a directory accessible to ColdFusion and your web server, and name it `helloExamples`. This directory can go under your web root directory.
- 2 Create a ColdFusion page, and save it as `helloWorld.cfm` in the `helloExamples` directory.
- 3 Edit the `helloWorld.cfm` page to insert the following code:

```
<cfset tempStruct = StructNew()>
<cfset tempStruct.timeVar = DateFormat(Now ())>
<cfset tempStruct.helloMessage = "Hello World">

<cfset Flash.Result = tempStruct>
```

In the example, you add two string variables to a structure, one with a formatted date and one with a simple message. The structure is returned to the Flash application using the `Flash.Result` variable.

- 4 Save the file.

The directory name is the service address, and the `helloWorld.cfm` file is a method of the `helloExamples` Flash Remoting service. The following ActionScript example calls the `helloWorld` ColdFusion page:

```
#include "NetServices.as"
NetServices.setDefaultGatewayUrl("http://localhost/flashservices/gateway");
gatewayConnection = NetServices.createGatewayConnection();
CFMService = gatewayConnection.getService("helloExamples", this);
CFMService.helloWorld();
```

Within the result handler of `helloWorld`, you access the structure returned by ColdFusion.

Returning record sets to Flash

One common reason for a Flash application to call a ColdFusion page is for the ColdFusion page to access a database and return a record set to the Flash application. For example, the following ColdFusion code executes a query and returns the results of the entire query to the Flash application:

```
<cfquery name="myQuery" datasource="ExampleApps">
    SELECT *
    FROM tblItems
</cfquery>
```

```
<cfset Flash.Result = myQuery>
```

You can pass parameters from the Flash application to the ColdFusion page to conditionalize the query. The previous section contained an example that passed an argument to the WHERE clause of the query, as the following code shows:

```
<cfquery name="myQuery" datasource="ExampleApps">
    SELECT ItemName, ItemDescription, ItemCost
    FROM tblItems
    WHERE ItemName='#{Flash.Params[1]}#'
</cfquery>
```

```
<cfset Flash.Result = myQuery>
```

Depending on the SQL code of the query and the amount of data stored in the database, the query can return a single record, a few records, or a very large number of records. To pass the entire record set to the Flash application, you only have to write the record set to the `Flash.Result` variable.

In ActionScript, you access the record set columns to display the query. For example, you use the following ActionScript to call a ColdFusion page named `cfQuery.cfm` that contains the previous query:

```
myService.cfQuery("RipperStik");
```

In the result handler for the `cfQuery()` function, you access the record set as follows:

```
function cfQuery_Result ( result )
{
    DataGlue.bindFormatStrings(employeeData, result,
        "#{ItemName}#",
        #ItemDescription#,
        #ItemCost#");
}
```

In this example, `employeeData` is a Flash list box. This result handler displays the columns `ItemName`, `ItemDescription`, and `ItemCost` from each record in the result set, separated by commas, in the list box.

Returning record sets in increments

ColdFusion lets you return record set results to Flash in increments. For example, if a query returns 20 records, you can set the `Flash.Pagesize` variable to return five records at a time to Flash. Incremental record sets let you minimize the time that the Flash application waits for the application server data to load.

The entire record set is called a **Pageable Record Set** and each increment is called a **page**. Therefore, the `Flash.Pagesize` variable sets the number of records in each page.

The ColdFusion page executes once and returns all the results to the Flash Remoting gateway. The Flash application then requests subsequent records from the gateway as required.

To create a ColdFusion page that returns an incremental record set to Flash:

1 Create a ColdFusion page, and save it as `getData.cfm` in the `helloExamples` directory.

2 Modify `getData.cfm` so that the code appears as follows:

```
<cfset Flash.Pagesize = Flash.Params[1]>
<cfquery name="myQuery" datasource="ExampleApps">
    SELECT ItemName, ItemDescription, ItemCost
    FROM tblItems
</cfquery>
<cfset Flash.Result = myQuery>
```

In this example, you pass a parameter from the Flash application that defines the increment size.

3 Save the file.

When you assign a value to the `Flash.Pagesize` variable, you specify that if the record set has more than that number of records, the record set becomes pageable and returns the number of records specified in the `Flash.Pagesize` variable. For example:

```
#include "NetServices.as"
NetServices.setDefaultGatewayUrl("http://localhost/flashservices/gateway");
gatewayConnection = NetServices.createGatewayConnection();
CFMService = gatewayConnection.getService("helloExamples", this);
CFMService.getData(10);
```

Flash UI components are designed to recognize results returned in increments. After the initial delivery of records, the `RecordSet` ActionScript class becomes responsible for fetching records. You can configure the client-side `RecordSet` object to fetch records in various ways using the `setDeliveryMode` ActionScript function.

In many cases, you do not have to modify the Flash UI components to handle data returned in increments. For example, if the record set is returned to a Flash list box, the list box requests more rows as the user scrolls through the list box.

Using Flash Remoting MX with ColdFusion components

ColdFusion components require little modification to work with Flash. The `cffunction` tag names the function and contains the application logic, and the `cfreturn` tag returns the result to Flash.

Determining the Flash service name

The service name in ActionScript corresponds to the name of the `.cfc` file that contains the ColdFusion component. For example, if you create a ColdFusion component in the file `flashComp.cfc`, and the file is located in the directory `helloExamples` under your web root directory, you would define the service in ActionScript as follows:

```
#include "NetServices.as"
NetServices.setDefaultGatewayUrl("http://localhost/flashservices/gateway");
gatewayConnection = NetServices.createGatewayConnection();
CFService = gatewayConnection.getService("helloExamples.flashComp", this);
```

Within ColdFusion components, you create functions that define the functionality of the component. After defining the ActionScript service, you call component functions directly from ActionScript. For example, the following component defines two functions:

```
<cfcomponent>
  <cffunction name="functA" access="remote" returnType="Struct">
    ...
  </cffunction>

  <cffunction name="functB" access="remote" returnType="Struct">
    ...
  </cffunction>
</cfcomponent>
```

Note: For ColdFusion component methods to communicate with Flash applications, you must set the `cffunction` tag's `access` attribute to `remote`.

You call these functions in ActionScript using the following syntax:

```
CFService.functA();
CFService.functB();
```

Returning results to ActionScript

In a ColdFusion component, you use the `cfreturn` tag to return a single variable to ActionScript. The following example returns a structure variable:

```
<cfcomponent>
  <cffunction name="helloWorld" access="remote" returnType="Struct" >
    ...
    <cfreturn tempStruct>
  </cffunction>
</cfcomponent>
```

Returning record sets in increments from a component

ColdFusion lets you return record set results to Flash in increments. For example, if a query returns 20 records, you can set the `Flash.Pagesize` variable to return five records at a time to Flash. Incremental record sets let you minimize the time that the Flash application waits for the application server data to load.

The component executes once and returns all the results to the Flash Remoting gateway. The Flash application then requests subsequent records from the gateway as required.

The following example sets the `Flash.Pagesize` variable to 10 as part of returning results to the Flash application:

```
<cfcomponent>
  <cffunction name="getQuery" access="remote" returnType="query" >
    <cfquery name="myQuery" datasource="ExampleApps">
      SELECT *
      FROM tblItems
    </cfquery>
    ...
    <cfset Flash.Pagesize = 10>
    <cfreturn myQuery>
  </cffunction>
</cfcomponent>
```

For more information, see “Returning record sets in increments” on page 76.

Passing parameters to ColdFusion components

You can pass multiple parameters from ActionScript to a ColdFusion component. In a component, you either include the `cfargument` tag within a function definition that corresponds to each parameter, or access the parameters directly from the Arguments scope.

The order in which you specify the `cfargument` tags in the function corresponds to the order in which the parameters are passed from ActionScript. For example, the following ActionScript call passes three parameters:

```
CFService.functA(a, b, c);
```

The corresponding ColdFusion component defines three arguments, including the data type of the parameter:

```
<cfcomponent>
  <cffunction ...>
    <cfargument name="arg_for_a" type="type_of_a">
    <cfargument name="arg_for_b" type="type_of_b">
    <cfargument name="arg_for_c" type="type_of_c">
    ...
  </cffunction ...>
</cfcomponent>
```

For information on how ActionScript data types are converted to ColdFusion data types, see Chapter 3, “Using Flash Remoting Data in ActionScript” on page 35.

The following table describes using the `cfargument` tag to access parameters:

Collection	ActionScript example	Notes
Strict array	<pre>var myArray = new Array(); myArray[0] = "zero"; myArray[1] = "one"; myService.myMethod(myArray);</pre>	<p>The Flash Remoting service converts the array argument to a ColdFusion MX array. All CFML array operations work as expected.</p> <pre><cffunction ...> <cfargument name="arg1" type="array"> <cfset p1=arg1[1]> <cfset p2=arg1[2]> ... </cffunction ...></pre>
Named or associative array	<pre>var myStruct = new Array(); myStruct["zero"] = "banana"; myStruct["one"] = "orange"; myService.myMethod(myStruct);</pre>	<p>In ActionScript, named array keys are not case-sensitive.</p> <pre><cffunction ...> <cfargument name="arg1" type="struct"> <cfset p1=arg1.zero> <cfset p2=arg1.one> ... </cffunction ...></pre>
Named arguments using object initializer	<pre>myService.myMethod({x:1, y:2, z:3});</pre>	<p>Provides a convenient way of passing named arguments to ColdFusion pages. Access these arguments as normal named arguments of a component function.</p> <pre><cffunction ...> <cfargument name="x" type="numeric"> <cfargument name="y" type="numeric"> <cfargument name="z" type="numeric"> <cfset p1=x> <cfset p2=y> <cfset p3=z> ... </cffunction ...></pre>

The following example replicates the `helloWorld` function that was previously implemented as a ColdFusion page. For more information, see “Using Flash Remoting MX with ColdFusion pages” on page 70.

To create a ColdFusion component that interacts with a Flash application:

- 1 Create a ColdFusion component, and save it as `flashComponent.cfc` in the `helloExamples` directory.
- 2 Edit `flashComponent.cfc` so that it appears as follows:

```
<cfcomponent>
  <cffunction name="helloWorld" access="remote" returnType="Struct">
    <cfset tempStruct = StructNew()>
    <cfset tempStruct.timeVar = DateFormat(Now ())>
    <cfset tempStruct.helloMessage = "Hello World">
    <cfreturn tempStruct>
  </cffunction>
</cfcomponent>
```

This example creates the helloWorld function. The cfreturn tag returns the result to the Flash application.

3 Save the file.

The following ActionScript example calls this function:

```
#include "NetServices.as"
NetServices.setDefaultGatewayUrl("http://localhost/flashservices/gateway");
gatewayConnection = NetServices.createGatewayConnection();
CFService = gatewayConnection.getService("helloExamples.flashComponent", this);
CFService.helloWorld();
```

For ColdFusion components, the component filename, including the directory structure from the web root, serves as the service name. Remember to use a period to delimit the path directories, rather than a backslash.

Accessing ActionScript objects

ActionScript supports the object initializer syntax when calling a function. For example, the following function call passes two parameters as objects:

```
myService.myMethod({x:1, y:2});
```

In this example, the function passes x with a value of 1 and y with a value of 2.

In your component, you can access objects using the object name, as in the following example:

```
<cfcomponent>
  <cffunction ...>
    <cfargument name="x" type="numeric">
    <cfargument name="y" type="numeric">
    ...
  </cffunction ...>
</cfcomponent>
```

You can also pass arrays, structures, and named objects using this syntax. The following ActionScript defines an object:

```
params = new Object();
params.first = "Hello";
params.second = true;
service.concat(params);
```

In a component, you access the object elements using named parameters, as follows:

```
<cfcomponent>
  <cffunction name="concat" access="remote" returntype="any">
    <cfargument name="first" type="any" required="true">
    <cfargument name="second" type="any" required="true">
    ...
    <cfreturn first & second>
  </cffunction>
</cfcomponent>
```

This component specifies that two parameters are required. An ActionScript object will satisfy this requirement, as it will be split into named arguments. However, an ActionScript array will not.

Passing objects from ActionScript lets you use the Arguments scope within a component function. The Arguments scope works the same away as the Flash scope in ColdFusion pages. In a component, you can access parameters using the syntax `Arguments.paramName`. Therefore, you can access the params object from the previous example, as follows:

```
<cfcomponent>
  <cffunction name="concat" access="remote" returnType="any">
    <cfset p1=Arguments.first>
    <cfset p2=Arguments.second>
    ...
  </cffunction>
</cfcomponent>
```

Using component metadata with the Flash Remoting service

Flash MX designers can use the Service Browser in the Flash MX authoring environment to discover business logic functionality built in ColdFusion. You use the `description` attribute of the `cffunction` and `cfargument` tags to describe the ColdFusion functionality to the Service Browser.

To create a ColdFusion component that describes itself to the Service Browser:

- 1 Edit `flashComponent.cfc` in the `helloExamples` directory to insert the following code:

```
<cfcomponent>
  <cffunction name="getTime" access="remote" returnType="Date"
    description="Returns date">
    <cfset tempStruct = StructNew(>
    <cfset tempStruct.timeVar = DateFormat(Now (>
    <cfreturn tempStruct>
  </cffunction>
  <cffunction name="sayHello" access="remote" returnType="Struct"
    description="Returns hello message">
    <cfset tempStruct = StructNew(>
    <cfset tempStruct.helloMessage = "Hello World">
    <cfreturn tempStruct>
  </cffunction>
</cfcomponent>
```

In this example, the `description` attribute of the `cffunction` tag supplies a short text description of the component method.

- 2 Save the file.
- 3 Open the Flash MX authoring environment, and open the Service Browser using the `Window > Service Browser` menu command.
- 4 If not already present, add the Flash Remoting gateway using your Flash Remoting service URL, such as `http://localhost/flashservices/gateway`.
- 5 Add the `flashComponent` service using the service address `helloExamples.flashComponent`.
- 6 Click the `getTime` or `sayHello` folder, the description appears in the Service Browser.

Using Flash Remoting MX with server-side ActionScript

The ability to create server-side ActionScript provides a familiar way for Flash developers to access ColdFusion query and HTTP features without learning CFML. You can place ActionScript files (.asr) that you want to call from the Flash application on the server, anywhere under the web server's root directory. To specify subdirectories of the webroot or a virtual directory, use package dot notation. For example, in the following assignment code, the `stockquotes.asr` file lives in the `mydir\stock\` directory:

```
stockService = gatewayConnection.getService("mydir.stock.stockquotes", this);
```

You can also point to virtual mappings, such as `cfsuite.asr.stock.stockquotes`, where `cfsuite` is a virtual mapping and `asr.stock` is a subdirectory of that mapping. The `CF.query` and `CF.http` functions give you a well-defined interface for building the SQL queries and HTTP operations of ColdFusion.

For example, the following server-side ActionScript function definition returns a `RecordSet` object:

```
function basicQuery()  
{  
    mydata = CF.query({datasource:"customers",  
        sql:"SELECT * FROM myTable"});  
    return mydata;  
}
```

Using CF.http

The `CF.http` ActionScript function lets you retrieve information from a remote HTTP server. HTTP `Get` and `Post` methods are supported. Using the `Get` method, you send information to the remote server directly in the URL. This method is often used for a one-way transaction in which the `CF.http` function retrieves an object, such as the contents of a web page. The `Post` method can pass variables to a form or CGI program, and can also create HTTP cookies.

One of the most basic and useful ways of using the `CF.http` function is using the `Get` method argument to retrieve a page from a specified URL. For example, the following server-side code retrieves file content from the specified URL:

```
function basicGet(url)  
{  
    // Invoke with just the url. This is an http get.  
    result = CF.http(url);  
    return result.get("Filecontent");  
}
```

In the client-side ActionScript, you call the service function and display the results in the Flash application, as in the following example:

```
#include "NetServices.as"  
if (inited == null)  
{  
    inited = true;  
    cfserver = NetServices.createGatewayConnection("http://localhost/  
        flashservices/gateway");  
    myHttpService = gatewayConnection.getService("httpFuncs", this);  
}
```

```

}
myHttpService.basicGet("http://www.macromedia.com");

function basicGet_Result(result)
{
    myDisplayScreen.text = result;
}

```

The `CF.http` function returns an object that contains properties (also known as attributes) that you reference to access the contents of the file returned, header information, HTTP status codes, and so on. The following table shows the properties available:

Property	Description
Text	A Boolean value that indicates whether the specified URL location contains text data.
Charset	The character set used by the document specified in the URL. HTTP servers normally provide this information, or the charset is specified in the <code>charset</code> parameter of the <code>Content-Type</code> header field of the HTTP protocol. For example, the following HTTP header announces that the character encoding is EUC-JP: Content-Type: text/html; charset=EUC-JP
Header	Raw response header. For example, <code>macromedia.com</code> returns the following header: HTTP/1.1 200 OK Date: Mon, 04 Mar 2002 17:27:44 GMT Server: Apache/1.3.22 (Unix) mod_perl/1.26 Set-Cookie: MM_cookie=207.22.48.162.4731015262864476; path=/; expires=Wed, 03-Mar-04 17:27:44 GMT; domain=.macromedia.com Connection: close Content-Type: text/html
Filecontent	File contents, for text and MIME files.
Mimetype	MIME type. Examples of MIME types include <code>text/html</code> , <code>image/png</code> , <code>image/gif</code> , <code>video/mpeg</code> , <code>text/css</code> , and <code>audio/basic</code> .
Responseheader	Response header. If there is one instance of a header key, you can access the value as a simple type. If there is more than one instance, values are put in an array in the <code>responseHeader</code> structure.
Statuscode	HTTP error code and associated error string, which returns the following HTTP status codes: 400: Bad Request 401: Unauthorized 403: Forbidden 404: Not Found 405: Method Not Allowed

The arguments in the following table can be passed only as an array of objects in the `params` argument of the `CF.http` function:

Argument	Description
name	Variable name for data that is passed
type	Transaction type: <ul style="list-style-type: none">• URL• FormField• Cookie• CGI• File
value	Value of URL, FormField, Cookie, File, or CGI variables that are passed

You can write the `CF.http` function using either named arguments or positional arguments. The positional argument style supports a subset of `CF.http` arguments, although the named argument style is more readable than the positional argument style.

The `CF.http` function accepts the following arguments using the named argument style:

```
CF.http
({
    method:"get or post",
    url:"URL",
    username:"username",
    password:"password",
    resolveurl:"yes or no",
    params:arrayvar,
    path:"path",
    file:"filename"
})
```

The named argument style uses curly braces to surround the function arguments. The positional argument approach supports a subset of `CF.http` arguments, but it lets you code in a more succinct and efficient style. The schema for the positional argument style is as follows:

```
CF.http(url);
CF.http(method, url);
CF.http(method, url, username, password);
CF.http(method, url, params, username, password);
```

When using positional arguments, do not use curly braces.

Using CF.query

The `CF.query` function lets you perform queries against any ColdFusion data source. The `CF.query` function maps closely to the `cfquery` CFML tag, although it currently supports a subset of the `cfquery` attributes.

You use the `CF.query` function to perform the following actions:

- Identify the data source you want to query
- Pass SQL statements to the data source
- Pass other optional parameters to the database

You can write the `CF.query` function using either named arguments or positional arguments. The named argument style is a more readable style than the positional argument style. Although the positional argument style supports a subset of `CF.query` arguments, it allows a more compact coding style that is appropriate for simple expressions of the `CF.query` function.

The `CF.query` function accepts the following arguments using the named argument style:

```
CF.query
({
    datasource:"data source name",
    sql:"SQL stmts",
    username:"username",
    password:"password",
    maxrows:number,
    timeout:milliseconds
})
```

The named argument style uses curly braces to surround the function arguments. The positional argument approach supports a subset of `CF.query` arguments, but it lets you code in a more succinct and efficient style. The schema for the positional argument style is as follows:

```
CF.query(datasource, sql);
CF.query(datasource, sql, maxrows);
CF.query(datasource, sql, username, password);
CF.query(datasource, sql, username, password, maxrows);
```

When using positional arguments, do not use curly braces.

The `CF.query` function returns a `RecordSet` object to Flash. For more information about working with `RecordSet` objects, see Chapter 3, “Using Flash Remoting Data in ActionScript” on page 35.

Calling web services from Flash Remoting MX

Using Flash Remoting MX with ColdFusion, you can interact with web services directly from your Flash applications without having to build a ColdFusion page or component. You write the code to reference a web service in your Flash application and use ColdFusion only to perform the access.

Web services are remote applications that expose their functions and associated parameters using the Web Services Description Language (WSDL). WSDL files describe the functionality of a web service, including available functions, parameters, and results. You use a Simple Object Access Protocol (SOAP) proxy to parse the WSDL and make the remote service functions available in your application.

ColdFusion MX contains a built-in SOAP proxy for interacting with web services. The ColdFusion Flash Remoting service also includes a web service adapter, which employs the ColdFusion SOAP proxy for calling methods, passing parameters, and returning results from web services. Using the ColdFusion SOAP proxy, much of the complexity associated with programming web services is removed.

Invoking web service methods using Flash Remoting MX

The following example shows the code you use to access a web service:

```
#include "NetServices.as"
if (inited == null)
{
    // do this code only once
    inited = true;

    NetServices.setDefaultGatewayUrl("http://localhost/flashservices/gateway");
    gateway_conn = NetServices.createGatewayConnection();
    myWebService = gateway_conn.getService("URL_to_WSDL", this);
}
```

The `setDefaultGatewayUrl` function specifies the Flash Remoting service URL in ColdFusion. The `getService` function creates a reference to the web service. No connection to ColdFusion is made until you make the service function call.

The following example `getService` function references a temperature web service, located at <http://www.xmethods.net/sd/2001/TemperatureService.wsdl>, which returns the local temperature by U.S. zip code:

```
myWebService = gateway_conn.getService("http://www.xmethods.net/sd/2001/
    TemperatureService.wsdl", this);
```

Once you create the connection, you can call methods of the web service.

The temperature web service contains a single method named `getTemp`. This method takes a string containing a zip code and returns the temperature as a float. The following code assumes that `inzip` represents an input text field:

```
myWebService.getTemp(inzip.text);
```

To handle the results of the web service method, you create an event handler with the same name as the service functions with `_Result` or `_Status` appended to the name. The result handler displays the results in the translate input text box, as the following example shows:

```
function getTemp_Result(result)
{
    tempDisplay.text = result;
}
function getTemp_Status(result)
{
    tempDisplay.text = error.description;
}
```

In this example, the result of the web service function call, represented by the result variable, is assigned to the text property of the text box `tempDisplay`, which displays in the Flash application.

Securing access to ColdFusion from Flash Remoting MX

You can control access to ColdFusion files from Flash using the ColdFusion security mechanism in the same way that you control access to any ColdFusion page. This allows you to grant Flash application access to only selected ColdFusion code.

ColdFusion security is based on a username and password. Flash Remoting applications can pass the username and password information using the `setCredentials` function in ActionScript. From within your ColdFusion Application.cfm page, you can use the `cflogin` tag to access this information.

The following example passes a username and password to ColdFusion:

```
if (inited == null)
{
    inited = true;
    NetServices.setDefaultGatewayUrl("http://localhost/flashservices/gateway");
    gatewayConnection = NetServices.createGatewayConnection();
    gatewayConnection.setCredentials("bob", "password");
    myService = gatewayConnection.getService("securityTest.theafc", this);
}
```

Note: Typically, you do not hard-code a username and password within a Flash application because .swf files can be easily decompiled.

You use the `cflogin` tag to retrieve the username and password information, as the following example Application.cfm file shows:

```
<cfsilent>
<cflogin>

    <cfif isDefined("cflogin")
        <!--- Verify user name from cflogin.name and password from cflogin.password
            using your authentication mechanism. For example, you might store this
            information in an LDAP database. --->
        >
    <cfif cflogin.name eq "bob">
        <!--- In this example, bob is in the role of administrator. Typically, you
            store user roles with authentication information. --->
        <cfloginuser name="#cflogin.name#" password="#cflogin.password#"
            roles="Admin">
    </cfif>

</cflogin>
</cfsilent>
```

This example does not show how to perform user verification. For more information on verification, see *Developing ColdFusion MX Applications with CFML*.

Assigning security roles to component functions

ColdFusion components offer roles-based security. The following example creates a component method that deletes files:

```
<cfcomponent>
  <cffunction name="deleteFile" access="remote" roles="admin,manager">
    <cfargument name="filepath" required="yes">
      <cffile action="DELETE" file=#arguments.filepath#>
    </cffunction>
</cfcomponent>
```

In the example, the `cffunction` tag includes the `roles` attribute to specify the user roles allowed to access it. In this example, only users in the `admin` and `manager` role can access the function. Multiple roles are delimited with a comma.

In the `Application.cfm` file, you use the `cfloginuser` tag to log in the user and assign the user to a role. The user must be assigned to the correct role to access the component function. For more information on roles, see *Developing ColdFusion MX Applications with CFML*.

Handling errors with ColdFusion

ColdFusion pages and components can return error information to a Flash application if the ColdFusion code fails. For example, the following Flash application calls a ColdFusion page named `causeError.cfm`:

```
#include "NetServices.as"
NetServices.setDefaultGatewayUrl("http://localhost/flashservices/gateway");
gatewayConnection = NetServices.createGatewayConnection();
CFMService = gatewayConnection.getService("errorExample", this);
CFMService.causeError();
```

To help with debugging your ColdFusion code, use the `cftry` and `cfcatch` tags in your ColdFusion page or component to catch errors and return helpful error messages about the errors to the Flash application. For example, the ColdFusion page `causeError.cfm` contains the following code:

```
<cftry>
  <cfset dev = Val(0)>
  <cfset Flash.Result = (1 / dev)>

  <cfcatch type="any">
    <cfthrow message="An error occurred in this service: #cfcatch.message#">
  </cfcatch>
</cftry>
```

In this example, the second `cfset` tag fails because it causes a divide-by-zero error. The `message` attribute of the `cfthrow` tag describes the error and is returned to the Flash application by ColdFusion. For more information on using the `cftry` and `cfcatch` tags, see *Developing ColdFusion MX Applications with CFML*.

To handle the error in your Flash application, you create a Status handler for the `causeError` method, as follows:

```
function causeError_Status ( error )
{
  resultBox.text = error.description;
}
```

In this example, `resultBox` is a Flash text box that displays the error message created by the `cfthrow` tag.

For more information on handling errors, see Chapter 2, “Using Flash Remoting Components in ActionScript” on page 13.

CHAPTER 6

Using Flash Remoting MX for Java

This chapter describes how to use Macromedia Flash Remoting MX with services running in Java application servers.

You can use Flash Remoting MX from ActionScript in a Flash application to call public methods on Java objects running in Java application servers. Flash Remoting MX supports the following types of Java objects:

- JavaBeans (stateful)
- Java classes (stateless)
- Enterprise JavaBeans (stateless session, stateful session, and entity beans)
- Java servlets and JSPs
- Java Management Extensions (JMX) MBeans; available in Macromedia JRun 4 only

Macromedia JRun 4 also lets you call functions on server-side ActionScript, which can in turn call methods on server-side Java objects.

Contents

- About Flash Remoting MX for Java 92
- Calling Java classes or JavaBeans from ActionScript 94
- Calling EJBs from Flash..... 100
- Calling servlets and JSPs from Flash..... 104
- Calling JMX MBeans from Flash (JRun only) 107
- Calling server-side ActionScript from Flash (JRun only) 109
- Handling function results in ActionScript 111
- Using Flash Remoting MX with JRun security 113
- Passing XML objects between Flash and Java 114
- Viewing Flash Remoting MX log entries..... 115

About Flash Remoting MX for Java

How does Flash Remoting MX for Java work?

Flash Remoting MX exposes Java objects as services that are accessible from Flash applications as ActionScript functions that correspond to Java object methods. A Flash developer writes ActionScript that uses the `NetServices` ActionScript class to connect to a remote Java application server, get a reference to a service, and invoke the service's functions.

To transport messages, Flash Remoting MX uses a binary message format called Action Message Format (AMF) delivered over HTTP and modeled on the Simple Object Access Protocol (SOAP) used in web services implementations. AMF is smaller and faster than standard SOAP, and is purely asynchronous and event-driven. It lets you send a variety of data types, including RecordSets, Java objects, primitives such as integers, Strings, XML documents, references to EJBOjects, and Dates across the wire. For more information about AMF, see "Understanding AMF," in Chapter 1.

The Flash Remoting gateway is a front controller on the Java application server that handles the conversion of data types between ActionScript and Java.

When the gateway receives a service request, the request passes through a set of filters that handle such things as serialization, logging, and security, before arriving at a **service adapter** designed to handle the service and invocation type. Flash Remoting MX has adapters specifically designed for JavaBeans, Java classes, EJBs, JMX MBeans, and server-side ActionScript.

Where does Flash Remoting MX fit into the Java application architecture?

A **design pattern** is a solution to a recurring problem. Many design patterns are used in the context of a model-view-controller architecture, in which you separate data access functionality from the user interface and control logic that uses that functionality.

In design pattern terminology, a Flash application that uses Flash Remoting MX is the view portion of an application, much like a JavaServer Pages (JSP)- or servlet-based front-end is. The Flash Player, running in a web browser or in stand-alone mode, is the client in which the view is rendered. The Flash Remoting gateway is a front controller that translates interactions with the Flash-based view into actions that server-side Java objects perform.

Using Flash Remoting MX, you can take advantage of common design patterns and frameworks in which relational data is mapped to objects such as EJBs, JavaBeans, value objects, JMX MBeans, or Java Collections returned to the presentation tier. Flash Remoting MX for Java relies on these patterns of data transfer, rather than working with relational data returned directly to Flash.

In addition to using built-in data type mapping, you can pass any serializable object from Java to a Flash client. The Java instance's fields, including private fields, are converted to ActionScript properties on an ActionScript result object; when working with JavaBeans, this feature gives you access to JavaBean properties.

Two design patterns, the **value object** and **session facade** patterns, can be particularly useful with Flash Remoting MX. Both patterns can reduce the number of remote method calls required in a Flash application.

You can use a value object to send a coarse-grained view of data to the server and get back fine-grained data. For example, you can call a single method on a value object that aggregates several method calls on an entity bean. The method result is returned to the Flash application as an ActionScript result object from which you can access data locally. This pattern can help reduce network traffic and response time, and reduce the load on EJBs. For more information about the value object pattern, go to http://java.sun.com/blueprints/patterns/j2ee_patterns/value_object/index.html.

You can use a session facade to provide a single point of contact to a set of EJBs. For example, you can call methods on a session bean (the session facade) that is capable of calling various methods on several other EJBs, depending on the user's current context in the application. This pattern reduces network traffic and makes it easier to support different types of clients, change the enterprise data model, or change the server implementation. For more information about the session facade pattern, go to http://java.sun.com/blueprints/patterns/j2ee_patterns/session_facade/index.html.

Calling Java classes or JavaBeans from ActionScript

This section describes how to call a JavaBean or Java class from Flash using Flash Remoting MX. There is only one significant difference in how Flash Remoting MX handles standard Java classes and JavaBeans. A Java class is stateless and a new object instance is created whenever a method is invoked. A JavaBean is stateful in the user's HTTP session. Using a JavaBean with Flash Remoting MX is similar to using the `jsp:useBean` tag in a JSP. Flash Remoting MX sends a JSESSIONID parameter to the Flash application, and NetServices appends the session ID value to all subsequent HTTP requests.

Note: The Flash Remoting session is independent of HttpSession objects available to JSPs and servlets. A stateful JavaBean instantiated through Flash Remoting MX cannot access an object stored in a session by a JSP or servlet. Conversely, a JSP or servlet cannot use its session to access a JavaBean instantiated through Flash Remoting MX.

Making a Java class or JavaBean available to Flash Remoting MX

To call a standard Java class or JavaBean with Flash Remoting MX, the class or bean must be available in the classpath of the Flash Remoting gateway. Unless the class or bean is in the same web application as the gateway, you typically add it to the system classpath.

The following table lists standard ways to add classes to the system classpath:

Application server	Classpath information
Sun™ ONE Web Server	In the Web Server Administration Server console, add classes to the Classpath field in the Configure JVM Attributes page of the Java panel.
IBM® WebSphere®	<p>In the WebSphere Application Server Console, add classes to the Classpath field of the JVM Settings page for your server. In the server-cfg.xml tree in the left pane of the console, JVM Settings is under WebSphere Administrative Domain > Nodes > nodename > Application Servers > servername > Process Definition.</p> <p>Note: To call a class or JavaBean in WebSphere, you also must grant clients permission to access the package that contains the class or JavaBean. To do this, you add a line to the default permissions granted to all domains in the websphere_root/AppServer/java/jre/lib/security/java.policy file. For example, the following line lets users access the Flash Remoting sample classes in the flashgateway.samples package:</p> <pre>permission java.lang.RuntimePermission "accessClassInPackage.flashgateway.samples";</pre>
JRun	Copy classes to the jrun_root/jrun_server/SERVER-INF/classes directory in subdirectories that match the package structure of the classes. Copy JAR files to the jrun_root/jrun_server/SERVER-INF/lib directory.

Getting a reference to a Java class or JavaBean in ActionScript

Before calling methods of a Java class or JavaBean from ActionScript, you must get a reference to the Java object.

To get a reference to a Java object:

- 1 Include the NetServices.as file:

```
#include "NetServices.as"
```

- 2 Specify the default Flash Remoting gateway URL:

```
NetServices.setDefaultGatewayUrl("http://localhost/flashservices/gateway");
```

Note: There are several other ways to specify the gateway URL. For more information, see “Configuring Flash Remoting MX,” in Chapter 2.

- 3 Connect to the Flash Remoting gateway:

```
gatewayConnection = NetServices.createGatewayConnection();
```

- 4 Get a reference to the Java class or JavaBean in the HTTP session, as shown in the following example:

```
flashtestService = gatewayConnection.getService  
("flashgateway.samples.FlashJavaBean", this);
```

The first parameter of the `getService` function is the fully qualified class name of the Java class or JavaBean. The second parameter of the `getService` function, `this`, specifies that the results of service function calls are returned to this Flash timeline.

Invoking Java methods in ActionScript

Once you have a reference to a Java class or JavaBean, you can use ActionScript functions to invoke that object’s public methods. For example, to invoke the following JavaBean method:

```
public String getMessage() {  
    count++;  
    return message + " (count=" + count + ")";  
}
```

You could use the following ActionScript code, assuming `flashtestService` represents your reference to the JavaBean:

```
function getMessage()  
{  
    flashtestService.getMessage();  
}
```

To handle the function results, you use a result handler function like the following:

```
function getMessage_Result( result )  
{  
    messageOutput.text = result;  
}
```

For more information about handling function results in ActionScript, see “Handling function results in ActionScript” on page 111.

Looking at a Flash application that calls a JavaBean

The following sections show the three pieces required to call a JavaBean from a Flash application that uses Flash Remoting MX:

- JavaBean
- Flash user interface
- ActionScript

Looking at the JavaBean

The example Flash application invokes the `setMessage`, `getMessage`, `testBoolean`, and `testDate` methods of the following JavaBean:

```
package com.samples;

import java.util.Date;
import java.io.Serializable;
import org.w3c.dom.Document;

public class FlashJavaBean
    implements Serializable {

    private String message;
    private int count;

    public FlashJavaBean() {
        message = "Hello World From JavaBean";
        count = 0;
    }

    public boolean testBoolean(boolean b) {
        return b;
    }

    public Date testDate(Date d) {
        return d;
    }

    public void setMessage(String message) {
        this.message = "Hi " + message;
    }

    public String getMessage() {
        count++;
        return message + " (count=" + count + ")";
    }

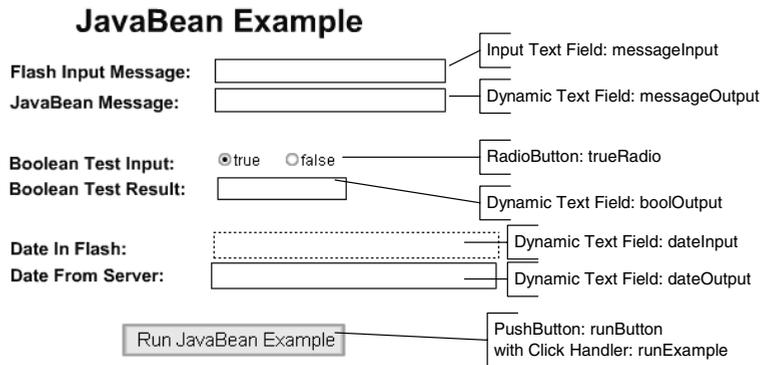
    public int getCount() {
        count++;
        return count;
    }

    public void setCount(int count) {
        this.count = count;
    }

    public Document testDocument(Document doc) {
        return doc;
    }
}
```

Looking at the user interface

The following figure shows the user interface of the example Flash application with callouts that indicate the field types and variable names referenced in the ActionScript code:



Looking at the ActionScript

The following code shows the ActionScript of the example Flash application, with comments in bold:

```
// Include NetServices library.  
#include "NetServices.as"  
#include "NetDebug.as"  
if (inited == null)  
{  
    // do this code only once  
    inited = true;  
  
// Set the default gateway URL.  
NetServices.setDefaultGatewayUrl("http://localhost/flashservices/gateway");  
  
// Connect to the gateway.  
gatewayConnection = NetServices.createGatewayConnection();  
// Get reference to JavaBean:  
flashtestService = gatewayConnection.getService  
("flashgateway.samples.FlashJavaBean", this);  
  
flashDate = new Date();  
  
// Set initial text for messageInput and dateInput text fields.  
messageInput.text = "[Enter a Message]";  
dateInput.text = "" + flashDate;  
}  
  
// Invoke business methods when user clicks the runButton.  
function runExample()  
{  
    setMessage();  
    getMessage();  
    testBoolean();  
}
```

```

        testDate();
    }

    // Business functions.
    function setMessage()
    {
        flashtestService.setMessage(messageInput.text);
    }

    function getMessage()
    {
        flashtestService.getMessage();
    }

    function testBoolean()
    {
        if (trueRadio.GetState())
        {
            flashtestService.testBoolean(true);
        }
        else
        {
            flashtestService.testBoolean(false);
        }
    }

    function testDate()
    {
        flashDate = new Date();
        dateInput.text = "" + flashDate;
        flashtestService.testDate(flashDate);
    }

    // Handle results from server; display results in output fields.
    function setMessage_Result( result )
    {
    }

    function getMessage_Result( result )
    {
        messageOutput.text = result;
    }

    function setMessage_Status( result )
    {
        messageOutput.text = result.details;
    }

    function getMessage_Status( result )
    {
        messageOutput.text = result.details;
    }

    function testBoolean_Result(result)
    {
        boolOutput.text = "result: " + result;
    }

```

```
}  
  
function testBoolean_Status(result)  
{  
    boolOutput.text = "status: " + result.details;  
}  
  
function testDate_Result(result)  
{  
    flashDate = result;  
    dateOutput.text = " " + flashDate;  
}  
  
function testDate_Status(result)  
{  
    dateOutput.text = "Status: " + result.details;  
}
```

Calling EJBs from Flash

The following sections describe how to get a reference to an EJBHome object and call EJB methods from ActionScript.

Getting a reference to an EJBHome object in ActionScript

Before calling the methods of an EJB from ActionScript, you must get a reference to an EJBHome object.

To get a reference to an EJBHome object:

- 1 Include the NetServices.as file:

```
#include "NetServices.as"
```
- 2 Specify the default Flash Remoting gateway URL:

```
NetServices.setDefaultGatewayUrl("http://localhost/flashservices/gateway");
```

Note: There are several other ways to specify the gateway URL. For more information, see “Configuring Flash Remoting MX,” in Chapter 2.
- 3 Connect to the Flash Remoting gateway:

```
gatewayConnection = NetServices.createGatewayConnection();
```
- 4 Get a reference to the EJBHome object; you must provide the JNDI name of the EJBHome object as the first parameter of the `getService` function, as shown in the following example where `SampleLoan` is the JNDI name:

```
SampleLoanHome = gatewayConnection.getService("SampleLoan", this);
```

The first parameter of the `getService` function is the JNDI name of the EJBHome object; the JNDI name cannot contain a period (.) character. The second parameter of the `getService` function, `this`, specifies that the results of service function calls are returned to this Flash timeline.

Invoking EJB methods in ActionScript

Unlike JavaBeans and Java classes, you must invoke the `create` method of an EJBHome object and return an EJBObject object before calling EJBObject methods. After you call the `create` method of an EJBHome object, you can use the ActionScript `create_Result(result)` function to get a reference to the EJBObject object and invoke its methods.

For example, to invoke the following method of a stateless session bean that performs loan calculations based on loan principal, term, and interest rate:

```
public double calculate(double principal, int months, float rate){
    if (rate < 0 || rate>1) return 0.0;
    double monthlyPayment = principal * (rate / (1 - Math.pow(1 +
        rate, -months)));
    return monthlyPayment;
}
```

You could use the following ActionScript code:

```
function runExample()
{
    SampleLoanHome = gatewayConnection.getService("SampleLoanEjbHome", this);
    SampleLoanHome.create();
}
function create_Result( result )
{
    flashStatelessEJB = result;
    calculate();
}
function calculate()
{
    flashStatelessEJB.calculate( (number (principalInput.text)), (number
        (monthsInput.text)), (number (rateInput.text)) );
}
}
```

To handle the function results, you use a result handler function like the following:

```
function calculate_Result (result)
{
    payOutput.text = result;
}
}
```

For more information about handling function results in ActionScript, see “Handling function results in ActionScript” on page 111.

Looking at a Flash application that calls an EJB

The following sections illustrate the three pieces required to call an EJB from Flash Remoting MX:

- EJB
- Flash user interface
- ActionScript

Looking at the EJB

The example Flash application calculates loan payments by invoking the calculate method on the following stateless session bean:

```
package ejbeans;

import java.rmi.*;
import java.util.*;
import javax.ejb.*;

public class SampleLoanBean implements SessionBean
{
    //////////////////////////////////////////////////
    // General Enterprise EJB stuff
    //////////////////////////////////////////////////
    private SessionContext sessionContext;
    public void ejbCreate() throws CreateException{};
    public void ejbRemove() throws RemoteException{};
    public void ejbActivate() throws RemoteException{};
}
```

```

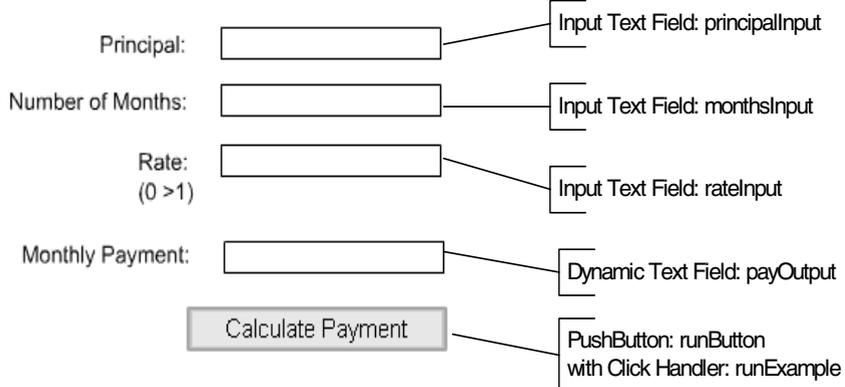
    public void ejbPassivate() throws RemoteException{};
    public void setSessionContext(SessionContext context) throws RemoteException {
        sessionContext = context;
    }
}
public double calculate(double principal, int months, float rate){
    if (rate < 0 || rate>1) return 0.0;
    double monthlyPayment = principal * (rate / (1 - Math.pow(1 +
        rate,-months)));
    // Double payMe=new Double (monthlyPayment);
    return monthlyPayment;
}
}
}

```

Looking at the user interface

The following figure shows the user interface of the example Flash application with callouts that indicate the field types and variable names referenced in the ActionScript code:

EJB example: loan calculator



Looking at the ActionScript

The following code is the ActionScript of the example Flash application:

```

// Include NetServices library.
#include "NetServices.as"
#include "NetDebug.as"
// -----
// Start up the application
// -----

if (inited == null)
{
    // do this code only once
    inited = true;

    // Set the default gateway URL.
    NetServices.setDefaultGatewayUrl("http://localhost/flashservices/gateway");
}

```

```

// Connect to the gateway.
gatewayConnection = NetServices.createGatewayConnection();
}

// -----
// Handlers for user interaction events
// -----
// Get reference to EJBHome and create EJBObject when user
// clicks the runButton.
function runExample()
{
flashstatelessHome = gatewayConnection.getService
("SampleLoanEjbHome", this);
flashstatelessHome.create();
}

// -----
// Business Methods
// -----
// Calculate payment based on data user enters in principalInput,
// monthsInput, and rateInput text fields. Function arguments must
// be numbers. Java method expects double, int, and float.
function calculate()
{
flashStatelessEJB.calculate( (number (principalInput.text)),
(number (monthsInput.text)), (number (rateInput.text)) );
}

// -----
// Handlers for data coming in from server
// -----
// Get reference to EJBObject and invoke calculatePayment function.
function create_Result( result )
{
flashStatelessEJB = result;

calculate();
}

// Get reference to calculate result object and display result
// in PayOutput text field.
function calculate_Result (result)
{
payOutput.text = result;
}

```

Calling servlets and JSPs from Flash

The following sections describe how to get a reference to a servlet or a JSP defined as a servlet in a `web.xml` file, and call the servlet or JSP.

Note: Servlets are supported on Servlet 2.2- and Servlet 2.3-compliant application servers. JSPs are only supported on Servlet 2.3-compliant application servers.

Coding a servlet to use with Flash Remoting MX

Although you can use any servlet with Flash Remoting MX, Flash Remoting MX provides a `FlashServlet` that subclasses the Servlet API and provides a better API than the Request scope. To use the `FlashServlet`, your servlet must be in the Flash Remoting web application. It must subclass the `flashgateway.adapter.java.FlashServlet` class and implement the following abstract method:

```
public Object service(ServletRequest req, ServletResponse resp, List arguments);
```

Getting a reference to a web application in ActionScript

Before calling a servlet or a JSP defined as a servlet, you must get a reference to the context root of the web application that contains the servlet or JSP.

To get a reference to a web application that contains a servlet or JSP:

- 1 Include the `NetServices.as` file:

```
#include "NetServices.as"
```

- 2 Specify the default Flash Remoting gateway URL:

```
NetServices.setDefaultGatewayUrl("http://localhost/flashservices/gateway");
```

Note: There are several other ways to specify the gateway URL. For more information, see “Configuring Flash Remoting MX,” in Chapter 2.

- 3 Connect to the Flash Remoting gateway:

```
gatewayConnection = NetServices.createGatewayConnection();
```

- 4 Get a reference to the context root of the web application that contains the servlet or JSP, as shown in the following example:

```
servletService = gatewayConnection.getService("mycontextroot", this);
```

The first parameter of the `getService` function must be the context root of the web application that contains the servlet or JSP. The second parameter of the `getService` function, `this`, specifies that the results of service function calls are returned to this Flash timeline.

Calling a servlet or JSP

To call a servlet or a JSP defined as a servlet from ActionScript, use the servlet name specified in the web application's web.xml deployment descriptor file as an ActionScript function name. For example, the servlet name is MyServlet in the following example:

```
function go_Clicked()
{
    servletService.MyServlet();
}
```

The web.xml file contains the following servlet definition:

```
<servlet>
  <servlet-name>MyServlet</servlet-name>
  <display-name>MyServlet</display-name>
  <description>Simple text servlet</description>
  <servlet-class>MyServlet</servlet-class>
</servlet>
```

Note: On Servlet 2.3-compliant application servers, you can define a JSP as a servlet by specifying a JSP file name in a jsp-file element, rather than a servlet class in a servlet-class element.

Request arguments sent from Flash as parameters of the *ServletName()* function are available from the Request scope as the parameter "FLASH.PARAMS". You can return results to Flash using the request parameter "FLASH.RESULT", as shown in the following servlet:

```
import javax.servlet.*;
import java.io.IOException;
import java.util.List;

public class MyServlet implements Servlet
{
    private String message = null;

    public void init(ServletConfig config) throws ServletException
    {
        message = "Hello from MyServlet";
    }
    public void service(ServletRequest request, ServletResponse response)
        throws ServletException, IOException
    {
        //The args could be used here too...
        /*
        Object o = request.getAttribute("FLASH.PARAMS");
        if (o instanceof List)
        {
            List args = (List)o;
        }
        Object arg0 = args.get(0);
        Object arg1 = args.get(1);
        */
        request.setAttribute("FLASH.RESULT", message);
    }
    public String getServletInfo()
    {
```

```
        return "A test servlet.";
    }
    public ServletConfig getServletConfig()
    {
        return null;
    }
    public void destroy()
    {
        message = null;
    }
}
```

To handle the function results in ActionScript, you use a result handler function like this one:

```
function MyServlet_Result (result)
{
    ResultBox.text = result;
}
```

For more information about handling function results in ActionScript, see “Handling function results in ActionScript” on page 111.

Calling JMX MBeans from Flash (JRun only)

You can call Macromedia JRun application functionality through JMX using Flash Remoting MX. You connect to a JMX MBean object using the MBean object name in the ActionScript `getService` function. Service functions are methods defined in the MBean's manageable interface. The following sections describe how to get a reference to a JMX MBean object and call its methods.

To grant clients access to JMX MBeans, you must provide permissions that specify the exposed MBean object names in the `jrun_root/lib/jrun.policy` file. For example, the following line of text, which is currently uncommented in the `jrun.policy` file, exposes the `DeployerService` for the JRun Flash samples:

```
permission jrun.security.JMXPermission
    "accessMBean.DefaultDomain:service=DeployerService";
```

You can also use wildcards to grant access to JMX MBeans. For example, to grant access to all MBeans under the `DefaultDomain` using a wildcard, you would uncomment the following line in the `jrun.policy` file:

```
// permission jrun.security.JMXPermission "accessMBean.DefaultDomain:*";
```

Getting a reference to an MBean in ActionScript

Before calling the methods of an MBean from ActionScript, you must get a reference to an MBean.

To get a reference to an MBean:

- 1 Include the `NetServices.as` file:

```
#include "NetServices.as"
```

- 2 Specify the default Flash Remoting gateway URL:

```
NetServices.setDefaultGatewayUrl("http://localhost/flashservices/gateway");
```

Note: There are several other ways to specify the gateway URL. For more information, see, "Using Flash Remoting Components in ActionScript," on page 13.

- 3 Connect to the Flash Remoting gateway:

```
gatewayConnection = NetServices.createGatewayConnection();
```

- 4 Get a reference to the MBean object; you must provide the name of the JMX object under which the MBean is registered in the `getService` function, as shown in the following example:

```
jrunDeployerMBean = gatewayConnection.getService ("DefaultDomain:service =
    DeployerService", this);
```

Invoking MBean methods in ActionScript

Once you have a reference to an MBean, you can use ActionScript functions to invoke its public methods. For example, you could use the following ActionScript code to invoke the `getEARS` method of the `jrunDeployerMBean` MBean to get a list of JMX object names for deployed enterprise applications:

```
function getEARS()
{
    jrunDeployerMBean.getEARS();
}
```

To handle the function results, you use a result handler function like the following:

```
function getEARS_Result(result)
{
    numDeployed.text = result.length;
}
```

For more information about handling results in ActionScript, see “Handling function results in ActionScript” on page 111.

Calling server-side ActionScript from Flash (JRun only)

This section describes how to call a server-side ActionScript file from Flash using Flash Remoting MX. The ability to create server-side ActionScript provides a familiar way for Flash developers to create server-side code without learning Java. You can also call server-side Java objects from server-side ActionScript. JRun uses the Mozilla Rhino JavaScript engine to support server-side ActionScript; for more information about calling Java objects with Rhino, go to <http://www.mozilla.org/rhino/scriptjava.html>.

Getting a reference to a server-side ActionScript file

Before calling the functions of a server-side ActionScript file (*.asr), you must get a reference to the file. Place the server-side ActionScript file that you want to call in a web application.

To get a reference to a server-side ActionScript file:

- 1 Include the NetServices.as file:

```
#include "NetServices.as"
```

- 2 Specify the default Flash Remoting gateway URL:

```
NetServices.setDefaultGatewayUrl("http://localhost/flashservices/gateway");
```

Note: There are several other ways to specify the gateway URL. For more information, see Chapter 2, “Using Flash Remoting Components in ActionScript” on page 13.

- 3 Connect to the Flash Remoting gateway:

```
gatewayConnection = NetServices.createGatewayConnection();
```

- 4 Get a reference to the server-side ActionScript file, as shown in the following example:

```
flashsasService = gatewayConnection.getService (flash.script.FlashSample",  
this);
```

The first parameter of the `getService` function is the directory structure and name of the ASR file relative to a web application context root. In this example, `flash` is the context root of the web application, `script` is a directory under the context root, and `FlashSample` is the prefix of the ASR file name. The second parameter of the `getService` function, `this`, specifies that the result of service function calls are returned to this Flash timeline.

Invoking server-side ActionScript functions

Once you have a reference to a server-side ActionScript file, you can use client-side ActionScript functions to call its functions. For example, to call the following server-side ActionScript function:

```
function getBiggerString(value)  
{  
    return "Hello from SSAS " + value;  
}
```

You could use the following client-side ActionScript code, assuming `flashssasService` represents your reference to the server-side ActionScript file:

```
function getBiggerString()
{
    flashssasService.getBiggerString( stringInput.text );
    //wait for the results
    stringOutput.text = "[Invoking SSAS...]";
}
```

To handle the function results, you use a result handler function like the following:

```
function getBiggerString_Result( result )
{
    stringOutput.text = result;
}
```

For more information about handling function results in ActionScript, see “Handling function results in ActionScript” on page 111.

Handling function results in ActionScript

Handling function results in ActionScript is the same for Java application servers and the other platforms that Flash Remoting MX supports. You write an ActionScript result handler function with a name composed of the name of the function returning the result with `_Result` appended to it. For example, to pass the return value of the following ActionScript function to ActionScript:

```
function calculate()
{
    loanService.calculate( (number (principalInput.text)), (number
        (monthsInput.text)), (number (rateInput.text)) );
}
```

You would use the following ActionScript function:

```
function calculate_Result( result )
{
    payOutput.text = result;
}
```

If a method returns an object that implements the Java `Serializable` interface, and all of its fields are serializable, its public and private properties are available as ActionScript properties. For example, the following method in a Java class called `Loan` calculates loan payments and returns an instance of a JavaBean called `LoanInfo` that stores principal, months, rate, and `monthlyPayment` property values:

```
public LoanInfo calculateReturnComplex(double principal, int months, float rate){
    if (rate < 0 || rate>1)
        principal = 0.0;
    double monthlyPayment = principal * (rate / (1 - Math.pow(1 + rate,-months)));
    LoanInfo info=new LoanInfo(principal, months, rate, monthlyPayment);
    return info;
}
```

You could use the following function to call this method in ActionScript:

```
function calculate()
{
    loanService.calculateReturnComplex( (number (principalInput.text)), (number
        (monthsInput.text)), (number (rateInput.text)) );
}
```

The `calculate` function returns the following `LoanInfo` bean:

```
package samples;
public class LoanInfo {
    private double principal;
    private int months;
    private float rate;
    private double monthlyPayment;
    public LoanInfo() {
    }
    public LoanInfo(double principal, int months, float rate, double monthlyPayment)
    {
        this.principal=principal;
        this.months=months;
    }
}
```

```

        this.rate=rate;
        this.monthlyPayment=monthlyPayment;
    }
    public LoanInfo(double principal, int months, float rate, double
        monthlyPayment, String message) {
        this.principal=principal;
        this.months=months;
        this.rate=rate;
        this.monthlyPayment=monthlyPayment;
        this.message=message;
    }
    public void setPrincipal(double principal) {
        this.principal=principal;
    }
    public double getPrincipal() {
        return principal;
    }
    public void setMonths(int months) {
        this.months=months;
    }
    public int getMonths() {
        return months;
    }
    public void setRate(float rate) {
        this.rate=rate;
    }
    public float getRate() {
        return rate;
    }
    public void setMonthlyPayment(double monthlyPayment) {
        this.monthlyPayment=monthlyPayment;
    }
    public double getMonthlyPayment() {
        return monthlyPayment;
    }
}

```

You can use the following ActionScript code to get the value of the LoanInfo bean's MonthlyPayment property:

```

function calculate_Result (result)
{
    payOutput.text = result.monthlyPayment;
}

```

For detailed information about the ActionScript result-handling hierarchy and result-handling strategies, see “Handling service results,” in Chapter 2.

Using Flash Remoting MX with JRun security

You can use the `NetServices.setCredentials` function in ActionScript to authenticate Flash users to a JRun 4 server and authorize them to access EJBs once they have been authenticated. When a user cannot be authenticated, the `NetServices.onStatus` function provides details on the client.

All of the gateway adapters support authentication, but only the EJB adapter supports authorization. Once a user has been authenticated, the user's role is associated with any EJBs the user may access, and the security permissions established in EJB deployment descriptors (`ejb-jar.xml`) are enforced. Only users in specified roles are allowed to access the EJB methods. An error occurs when the credentials provided through the `setCredentials` function do not map to a user in a specified role.

The following sections show examples of ActionScript for setting credentials and security settings.

Looking at the ActionScript

The following example shows ActionScript code that passes a username and password from a Flash application to a JRun server:

```
gatewayConnection = NetServices.createGatewayConnection();
gatewayConnection.setCredentials("Flash", "Flashpass");
```

Looking at the JRun security settings

The following example shows entries for the user named Flash and a role, FlashRole, to which the user belongs, using the default security implementation (`jrun-users.xml` file) for a JRun server:

```
<user>
  <user-name>Flash</user-name>
  <password>Flashpass</password>
</user>
<role>
  <role-name>FlashRole</role-name>
  <user-name>Flash</user-name>
</role>
```

The following example shows entries for the FlashRole role and corresponding EJB method permissions in an `ejb-jar.xml` file:

```
<assembly-descriptor>
  <security-role>
    <role-name>FlashRole</role-name>
  </security-role>
  <method-permission>
    <role-name>FlashRole</role-name>
    <method>
      <ejb-name>SampleLoanBean</ejb-name>
      <method-name>calculate</method-name>
    </method>
  </method-permission>
</assembly-descriptor>
```

Passing XML objects between Flash and Java

Both ActionScript and Java have object types for storing XML documents, and you can use Flash Remoting MX to send XML documents back and forth between ActionScript and Java.

In Flash, you can use an ActionScript XML object to represent and manipulate XML an document tree. In Java, you can use an `org.w3c.dom.Document` object to represent and manipulate an XML document tree. Flash Remoting MX converts ActionScript XML objects to `org.w3c.dom.Document` objects and `org.w3c.dom.Document` objects to ActionScript XML objects. For more information about ActionScript XML objects, see the Flash MX documentation set. For more information about the Java `org.w3c.dom.Document` interface, see the Java 2 Platform Standard Edition API documentation available at <http://java.sun.com>.

The following sections describe how you pass XML between ActionScript and Java.

Note: If you make a reference to an XML Document Type Definition (DTD) or schema in your XML document, you must use a Uniform Resource Identifier (URI) that the server can reach. Do not use a relative path.

Sending an ActionScript XML object to Java

The following ActionScript function creates an XML object and sends it as a parameter to a service function, which in this case is a JavaBean method that expects an `org.w3c.dom.Document` object; the first element of the XML document contains text entered in a Flash text field called `input`:

```
function testDocument()
{
    xmlDocument = new XML();

    firstElement = xmlDocument.createElement("TEST");
    firstElement.attributes.message = input.text;
    secondElement = xmlDocument.createElement("INSIDETEST");
    firstElement.appendChild(secondElement);

    xmlDocument.appendChild(firstElement);

    flashtestService.testDocument(xmlDocument);
}
```

Note: You can also create an XML document object in ActionScript by passing a string representation of the XML to the `new XML(source)` constructor. For more information, see the Flash MX documentation set.

Returning an XML object from Java to Flash

The JavaBean method discussed in the previous section returns an `org.w3c.dom.Document` object as follows:

```
public Document testDocument(Document doc)
{
    return doc;
}
```

Flash Remoting MX converts the returned `org.w3c.doc.Document` object to an ActionScript result of type XML object, which can be used in a result handler function as follows:

```
function testDocument_Result(result)
{
    output.text = result.firstChild.attributes["message"];
}
```

When the function above is called, the text contained in the first child element of the XML document root is displayed in a Flash dynamic text field called `output`. The text matches whatever text the user initially entered in the Flash text field called `input`.

Viewing Flash Remoting MX log entries

Flash Remoting MX writes messages to standard out and standard error, which appear in the following log files:

Application server	Log location
Sun One Web Server	error file in the <code>sunone_root/Servers/https-server_name/logs</code> directory
WebSphere	<code>servername_stdout</code> and <code>servername_stderr</code> files in the <code>websphere_root/AppServer/logs</code> directory
JRun	<code>servername_event</code> files in <code>jrun_root/logs</code> directory

Flash Remoting MX uses a default logger on all Java application servers other than JRun. You can set the default logger's logging level using the `LOG_LEVEL` context-param in the `web.xml` file of the Flash Remoting web application. Valid values are `None`, `Error`, `Warning`, `Information`, and `Debug`. By default, only errors are logged. The values are not case sensitive.

CHAPTER 7

Using Flash Remoting MX for Microsoft .NET

Macromedia Flash Remoting MX for Microsoft .NET is an ASP.NET web application that lets Macromedia Flash applications access and invoke ASP.NET pages, ADO.NET data, web services, and assemblies from ActionScript.

Contents

- About using Flash Remoting MX for Microsoft .NET 118
- Calling ASP.NET pages from Flash..... 122
- Using ADO.NET objects with Flash Remoting MX..... 128
- Calling web services from Flash 131
- Calling ASP.NET assemblies from Flash 134
- Viewing Flash Remoting log entries..... 137

About using Flash Remoting MX for Microsoft .NET

Macromedia Flash Remoting MX exposes ASP.NET technologies as **remote services** to Flash applications, which are accessible as ActionScript functions. A variety of Microsoft .NET technologies can serve as remote services, including ASP.NET pages, web services, and assembly methods. A Flash developer writes ActionScript that uses a library of functions called **NetServices** to connect to a remote .NET server, get a reference to the remote service, and invoke the remote service's functions.

To transport messages, Flash Remoting MX uses a binary message format called Action Message Format (AMF) delivered over HTTP and modeled on the Simple Object Access Protocol (SOAP) used in web services implementations. AMF is smaller and faster than standard SOAP, and is purely asynchronous and event-driven. It lets you send a variety of data types, including RecordSets, primitives like integers, strings, XML documents, and dates across the Internet using HTTP.

The Flash Remoting gateway acts as a front controller on the ASP.NET runtime that handles the conversion of data types from ActionScript to the .NET Common Language Runtime (CLR) and so on. When the gateway receives a service request, the request passes through a set of filters that handle serialization, logging, and security before arriving at a **service adapter** designed to handle the service and invocation type.

Flash Remoting MX contains four service adapters for .NET technologies:

- ASP.NET adapter
- ADO.NET data binding adapter
- Web services adapter
- Assembly (*.dll)

Flash Remoting MX for .NET requires the .NET Framework SDK to be installed. To check if you have the .NET Framework installed, open to the Windows Control Panel and double-click the Add/Remove Programs icon. In the Add/Remove Programs dialog box, look for Microsoft .NET Framework SDK. If you do not see it, go to the Microsoft website to download the SDK at <http://msdn.microsoft.com/downloads/default.asp?url=/downloads/sample.asp?url=/msdn-files/027/000/976/msdncompositedoc.xml>.

Where does Flash Remoting MX fit into the Microsoft .NET framework?

When embedded in ASPX pages with other server controls that render HTML, a Flash application that uses Flash Remoting MX becomes part of the client tier of a .NET application. Flash Remoting MX can be used as a custom server control in ASP.NET pages in .NET Web Form applications, or as a namespace in .NET DLL files, code-behind class files, and web services. A .NET assembly (flashgateway.dll), located within the local assembly cache of your ASP.NET application, provides the Flash Remoting functionality.

To assist you in planning your Flash applications, a **design pattern** represents a solution to a frequently experienced problem and provides a way to standardize coding practices across a complex project. Many design patterns are used in the context of a

model-view-controller architecture, in which you separate data access functionality from the user interface and control logic that uses that functionality.

Like an ASPX page, a Flash application represents the view portion. The Flash MX Player, running in a web browser or in stand-alone mode, is the client in which the view is rendered. The Flash Remoting gateway is a front controller that translates interactions with the Flash-based view into actions that server-side .NET technologies perform.

Two design patterns, the **Value Object** and **Session Facade** patterns, can be particularly useful with Flash Remoting MX. Both patterns can reduce the number of remote method calls required in a Flash application.

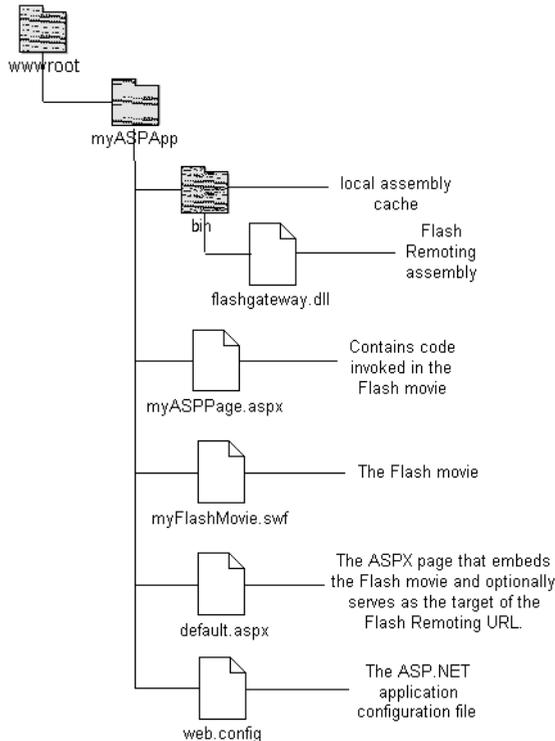
You can use a value object to send a coarse-grained view of data to the server and get back fine-grained data. For example, you can call a single method on a value object that aggregates several method calls on several different web services. The method result is returned to the Flash application as an ActionScript result object from which you can access data locally. This pattern can help reduce network traffic and response time.

The facade structural design pattern can be particularly useful with Flash Remoting MX by reducing the number of remote method calls required in a Flash application. You can use a service facade to provide a single point of contact to a set of ASPX pages or DLL methods. You can call methods on a DLL facade that is capable of calling various methods on several other DLLs, ASPX pages, or web services, depending on the user's current context in the application. This pattern reduces network traffic and makes it easier to support different types of clients, change the enterprise data model, or change the server implementation.

For more information on the .NET architecture and design patterns, go to <http://msdn.microsoft.com/architecture/>.

Understanding the Flash Remoting MX for Microsoft .NET directory structure

To enable an ASP.NET application with Flash Remoting MX, you place the Flash Remoting assembly in the application's local assembly cache (bin directory) and register the assembly in the application's web.config file. In the simplest form, your ASP.NET application directory structure might look like the following figure:



As the figure shows, the local assembly cache in the myASPApp directory contains the Flash Remoting assembly, flashgateway.dll. Also, to connect a Flash application with a remote service, you must reference a physical resource in your application's directory in the web root. Flash Remoting MX automatically installs a blank ASPX page named gateway.aspx, but you can use any ASPX file in the application directory.

The web.config file contains the registration for the flashgateway.dll. The HttpModule reference handles all web requests to ASP.NET resources. Here is an example:

```
<httpModules>
  <add name="GatewayController"
        type="FlashGateway.Controller.GatewayController,flashgateway" />
</httpModules>
```

If the request contains AMF, Flash Remoting MX proceeds with the request.

Note: The Flash Remoting MX for .NET installer creates a directory in your webroot named flashremoting. Inside the flashremoting directory, you will find a local assembly cache (bin directory) and a Samples directory, which contains example Flash applications.

Setting up a Flash Remoting-enabled ASP.NET application

Before you can start development with Flash Remoting MX, you must set up your ASP.NET application. The Flash Remoting MX for .NET installer creates a preconfigured application for you named flashremoting. You can find the flashremoting folder in your IIS webroot, such as C:/Inetpub/wwwroot/flashremoting.

When creating the directory structure for your own ASP.NET application that uses Flash Remoting MX, you must assign specific security permissions to the local assembly cache. Using Windows 2000 Professional as the operating system, the following procedure creates an ASP.NET application directory for Flash Remoting MX:

- 1 In your webroot, create a new folder named myASPApp.
- 2 In the myASPApp folder, create a new folder named bin. The bin folder serves as the local assembly cache.
- 3 Copy the Flash Remoting assembly, flashgateway.dll, from the installation directory to the bin folder in the myASPApp directory.
- 4 Using your mouse, right-click the bin folder. In the submenu that appears, select Properties. In the Properties dialog box, select the Security tab. In the Security panel, add the ASPNET user and give it Write access. Click OK.
- 5 In the myASPApp folder, create a blank ASPX page. The gateway URL points to this page.
- 6 Copy the web.config file from the flashremoting directory to the myASPApp directory. For Flash Remoting MX, the most important part of the web.config file is the `httpModule` tag that registers the Flash Remoting assembly, as the following example shows:

```
<httpModules>
  <add name="GatewayController"
        type="FlashGateway.Controller.GatewayController,flashgateway" />
</httpModules>
```

Your Flash Remoting-enabled ASP.NET application directory is now ready to use.

Calling ASP.NET pages from Flash

This section describes calling an ASPX page from Flash using Flash Remoting MX, working with the Flash Remoting custom server control in ASPX pages, using the Flash Remoting namespace in code-behind files, and so on. For more information on specific topics, see the following sections:

- Invoking ASPX pages in ActionScript
- Using the Flash Remoting custom server control in ASPX pages
- Using the Flash Remoting namespace in code-behind files
- Using ASP.NET state management with Flash Remoting MX
- Using ASP.NET exception handling

Making an ASP.NET page available to Flash Remoting MX

To call an ASP.NET (*.aspx) page from Flash Remoting MX, the ASPX page must reside within the directory or subdirectories of a Flash Remoting-enabled .NET application in the webroot.

Getting a reference to an ASPX-based service in ActionScript

Before calling an ASPX-based service from ActionScript in a Flash application, you must get a reference to the page.

To get a reference to the ASPX page:

- 1 Include the NetServices.as file:

```
#include "NetServices.as"
```

- 2 Specify the default Flash Remoting gateway URL:

```
NetServices.setDefaultGatewayUrl("http://localhost/myASPApp/default.aspx");
```

The gateway URL must reference an ASPX page inside the application directory. The `setDefaultGatewayURL` function should only be used during development in the Flash MX authoring environment. When you deploy the Flash application, you should supply the gateway URL using a parameter in the HTML that embeds the SWF file in the web page.

- 3 Connect to the Flash Remoting gateway:

```
gatewayConnection = NetServices.createGatewayConnection();
```

- 4 Get a reference to the ASPX page; you must provide the fully qualified path to the directory that contains the page to invoke, as shown in the following example:

```
ASPXservice = gatewayConnection.getService("myASPApp", this);
```

The first parameter specifies the directory that contains the ASPX page. The second parameter of the `getService` function, `this`, specifies that the result of service function calls are returned to this Flash timeline. For more information, see Chapter 2, “Handling service results” on page 27.

Invoking ASPX pages in ActionScript

Once you have a reference to the ASPX page, you can use ActionScript functions to invoke it. For example, the following ActionScript code invokes the ASPX page `myASPPage.aspx`, assuming that `ASPXservice` represents your reference to the directory that contains the ASPX page:

```
function getASPXPage()  
{  
    ASPXservice.myASPPage();  
}
```

The ASPX page's file name, `myASPPage.aspx`, becomes the function name, `myASPPage`, in the reference to the page's directory structure.

Using the Flash Remoting custom server control in ASPX pages

To access data passed from or return results to Flash applications in ASPX pages, you use the Flash Remoting custom server control in your ASPX page. The Flash Remoting server control is provided by the `flashgateway` DLL, which is located in the local assembly cache (bin directory) of your application. Like any custom server control, you must first register it in your ASPX page, as the following example shows:

```
<%@ Register TagPrefix="Macromedia" Namespace="FlashGateway"  
    Assembly="flashgateway" %>
```

The `Register` directive establishes the tag prefix (`Macromedia`), namespace (`FlashGateway`), and the assembly that provides the functionality (`flashgateway`). After you register the custom server control in your ASPX page, you can use it to pass data to Flash applications, as the following example shows:

```
<Macromedia:Flash ID="Flash" runat="server">  
    Hello from .NET!  
</Macromedia:Flash>
```

When invoked from the Flash application, the string `Hello from .NET!` is returned.

In addition to passing simple strings, you can write code in a .NET-supported language that accesses parameters passed from Flash and returns processed results to Flash. The Flash Remoting custom server control contains two properties for accessing passed parameters and returning results: `Flash.Params` and `Flash.Result`.

The `Flash.Params` property is a list consisting of parameters passed from a Flash application. The parameters arrive in the order that they were passed from the service function call in the ActionScript code of a Flash application. The `Flash.Result` property returns its value to Flash.

You can access Flash parameters like any other value in .NET, as the following C# example shows:

```
<%@ Page Language="C#" debug="true" %>  
<%@ Register TagPrefix="Macromedia" Namespace="FlashGateway"  
    Assembly="flashgateway" %>  
<Macromedia:Flash ID="Flash" Runat="Server" />  
<%  
    String message = "Hi ";  
    if (Flash.Params.Count > 0)
```

```

    {
        message += Flash.Params[0].ToString();
    }
    Flash.Result = message;
%>

```

Between the rendering blocks (<%...%>), the `if` statement condition, `Flash.Params.Count > 0`, evaluates the `Flash.Params` list for the number of parameters present. If a parameter is present, the parameter value, as a string, is appended to the `message` variable. Finally, the `message` variable is assigned into the `Flash.Result` property, which is returned to Flash.

If more than one parameter is passed from Flash, you access the parameters in your .NET application in the same order that they were passed from the Flash application. For example, the following ActionScript function passes two parameters, assuming `firstname` and `lastname` are input text fields in a Flash application:

```
ASPXservice.myASPPage(firstname.text, lastname.text);
```

In an ASPX page, for example, you access the parameters using strict array syntax, as the following VB.NET code shows:

```

<%@ Page language="vb" debug="true" CodeBehind="myASPPage.aspx.vb"
    AutoEventWireup="false" Inherits="myASPApp.myASPPage" %>
<%@ Register TagPrefix="Macromedia" Namespace="FlashGateway"
    Assembly="flashgateway" %>
<Macromedia:Flash ID="Flash" Runat="Server" />
<%
    dim message as string
    message = "Hi "
    if Flash.Params.Count > 0 then
        message = message & Flash.Params(0).ToString() & " " &
            Flash.Params(1).ToString()
    end if
    Flash.Result = message
%>

```

In the code, the `Flash.Params(0)` property represents the `firstname` parameter, and the `Flash.Params(1)` variable represents the `lastname` parameter. The Page directive references a code-behind file, `myASPPage.aspx.vb`.

Using the Flash Remoting namespace in code-behind files

In ASP.NET applications, you can separate business logic from user interface code using code-behind files. In the code-behind files, you use the Flash Remoting namespace to access parameters from and return results to Flash. To use code-behind files, you use the `codebehind` property of the page directive in an ASPX page, as the following example shows:

```

<%@ Page Language="c#" Debug="true" codebehind="myASPPage.aspx.cs"
    autoeventwireup="false" Inherits="myASPApp.myASPPage" %>
<%@ Register TagPrefix="Macromedia" Namespace="FlashGateway"
    Assembly="flashgateway" %>
<MACROMEDIA:FLASH id="Flash" Runat="Server" />

```

In the example, the page directive references the code-behind file in the `codebehind` property. The fully qualified class name is used in the `Inherits` property to inherit the methods of the code-behind file. You must also use the `register` directive to register the Flash Remoting custom server control, and then use the server control in the page.

In the code-behind file itself, you declare the Flash namespace as a protected variable in the class definition, as the following C# example shows:

```
namespace myASPApp
{
    public class myASPPPage : System.Web.UI.Page
    {
        protected FlashGateway.Flash Flash;
        ...
    }
}
```

The following VB.NET example performs the same operation:

```
Namespace Samples.asp
    Public Class CustomerInfo
        Inherits System.Web.UI.Page
        Protected Flash As FlashGateway.Flash
    End Class
End Namespace
```

After you establish the Flash Remoting namespace, you can manipulate the server control properties in the `Page_Load` method definition, as the following VB example shows:

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Handles MyBase.Load
    dim message as string
    message = "Hi "
    if Flash.Params.Count > 0 then
        message = message & Flash.Params(0).ToString() & " " &
            Flash.Params(1).ToString()
    end if
    Flash.Result = message
End Sub
```

In the code, if the `Flash.Params.Count` property is at least one, two parameters passed from Flash are appended to a string and returned to Flash.

Using ASP.NET state management with Flash Remoting MX

Flash Remoting MX supports ASP.NET cookie-based state management, which maintains a user session using HTTP header information. Flash Remoting MX maintains the session ID automatically by passing the session ID to the server in each subsequent service function request. To access session variables in Flash applications using Flash Remoting MX, you can code a service function that returns session variables using the `Flash.Result` property.

Note: Flash Remoting MX does not support .NET cookieless state management.

Setting and getting session variables with Flash Remoting MX

Using the Flash Remoting custom control properties, you can access and set the values of session variables. Session variables persist during a browser session from page to page. If you use Flash Remoting MX to return session variables, Flash movies embedded in different pages can access a common set of data.

To enable state management in an ASP.NET application, you use the `stateManagement` tag in the `config.web` file, as the following example shows:

```
<configuration>
  <sessionstate
    mode="inproc"
    cookieless="false"
    timeout="20"/>
</configuration>
```

To return a session variable, you use the `Flash.Result` property, as the following example shows:

```
<%@ Page language="c#" debug="true" %>
<%@ Register TagPrefix="Macromedia" Namespace="FlashGateway"
  Assembly="flashgateway" %>
<Macromedia:Flash ID="Flash" Runat="Server" />
<%
  Flash.Result = session.myPreference;
%>
```

In the code, the value of the `myPreference` variable is assigned in the `Flash.Result` property, which is returned to Flash. To set a session variable using a variable passed from Flash Remoting MX, you use the `Flash.Params` property, as the following example shows:

```
<%@ Page language="c#" debug="true" %>
<%@ Register TagPrefix="Macromedia" Namespace="FlashGateway"
  Assembly="flashgateway" %>
<Macromedia:Flash ID="Flash" Runat="Server" />
<%
if (Flash.Params.Count > 0)
{
  session.myPreference = Flash.Params[0].ToString();
}
%>
```

In the code, the parameter passed from Flash is assigned into the `myPreference` session variable.

Using ASP.NET exception handling

To return custom ASP.NET exceptions to Flash, you use the `throw` statement. You can throw exceptions in the context of a `try/catch` statement, an `if/else` statement, and so on. For example, the following C# snippet throws an exception:

```
if (Flash.Params.Count == 0)
{
    throw new Exception("No arguments received.");
}
```

In the code, if the `Flash.Params.Count` variable is zero, an exception is thrown. The exception message returns to Flash as part of the `onStatus` error object. To display the exception in Flash, you use the `error.description` property, as the following ActionScript snippet shows:

```
function serviceFunctionName_Status (result)
{
    textField = error.description;
}
```

In the code, the `error.description` property is assigned into the `textField` variable, which represents a text field in the Flash application. If you want to display the stack trace information returned from .NET, use the `error.details` property.

Using ADO.NET objects with Flash Remoting MX

Flash Remoting MX provides a service adapter for binding ADO.NET Data Tables and Data Views to the Flash Remoting custom server control. To bind data sets to the custom server control, you use the control's DataSource property and DataBind method. In ActionScript, the results are exposed as a RecordSet object.

The following C# example could be used in a code-behind file or in the ASPX page that contains the Flash Remoting server control, as the following example shows:

```
<%@ Page Language="c#" Debug="true" %>
<%@ Register TagPrefix="Macromedia" Namespace="FlashGateway"
    Assembly="FlashGateway" %>
<Macromedia:Flash id="Flash" Runat="Server" />
<%
    //create a SQL connection object and open a connection
    String source1 = "server=(local)\NetSDK;" + "id=QUser;pwd=QSPassword;" +
        "database=Northwind";
    SqlConnection sqlConnection = new SqlConnection(source1);
    sqlConnection.Open();

    //create the SQL statement
    String selectCountry = "SELECT DISTINCT Country FROM Customers ORDER BY Country
        ASC";

    //query the database
    SqlDataAdapter countryAdapter = new SqlDataAdapter(selectCountry,
        sqlConnection);

    //create a dataset object
    DataSet countryData = new DataSet();

    //fill the dataset with the query results
    countryAdapter.Fill(countryData, "Customers");

    //assign the dataset into the flash.datasource property
    Flash.DataSource = countryData.Tables["Customers"];

    //bind the datatable to the custom server control
    Flash.DataBind();

    //close the SQL connection
    sqlConnection.Close();
%>
```

In the code, the countryData DataSet is created from a SQL query to a database. Next, the countryData DataSet object is assigned into the Flash.DataSource property. Finally, the DataSet object is bound to the Flash Remoting custom server control using the Flash.DataBind method.

Data Tables are serialized by Flash Remoting MX to a RecordSet in ActionScript. Data Sets, which are collections of Data Tables, are serialized by Flash Remoting MX to an associative array of RecordSets back in ActionScript.

The following Visual Basic .NET example shows the Page_Load method definition in a code-behind file that performs the same operation as the previous code example:

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Handles MyBase.Load

    'create a SQL connection object and open a connection
    Dim source1 As String
    source1 = "Provider=Microsoft.Jet.OLEDB.4.0;Data
        Source=C:\\inetpub\\wwwroot\\flashremoting\\Samples\\ado\\Northwind.mdb;"
    sqlConnection = New OleDb.OleDbConnection()
    sqlConnection.ConnectionString = source1
    sqlConnection.Open()
    'create the SQL statement
    Dim selectAll As String
    selectAll = "SELECT ContactName, City, Phone, Country FROM Customers"

    'Initialize a DataSet contactData and string selectContactData
    Dim contactData As New DataSet()
    Dim selectContactData As String

    'Check for parameters from Flash
    If (Flash.Params.Count = 0) Then
        'create the SQL statement
        Dim selectCountry As String
        selectContactData = "SELECT DISTINCT Country FROM Customers ORDER BY Country
            ASC"
    Else
        Dim selectedCountryName As String
        'assign parameter passed from Flash to variable
        selectedCountryName = Flash.Params(0).ToString()
        Dim selectContactData As String
        'insert Flash parameter into SQL statement
        selectContactData = "SELECT ContactName, City, Phone FROM Customers WHERE
            Country = '\" + selectedCountryName + '\""
    End If

    'create the data adapter object
    Dim countryAdapter As System.Data.OleDb.OleDbDataAdapter

    'create a dataset object
    countryAdapter = New System.Data.OleDb.OleDbDataAdapter(selectAll,
        sqlConnection)

    'fill the dataset with the query results
    countryAdapter.Fill(contactData, "Customers")

    'assign the dataset into the flash.datasource property
    Flash.DataSource = contactData.Tables("Customers")

    'bind the datatable to the custom server control
    Flash.DataBind()

    'close the SQL connection
    sqlConnection.Close()
End Sub
```

In the code, the `contactData DataSet` is created from a SQL query to a database. Next, the `contactData DataSet` object is assigned into the `Flash.DataSource` property. Finally, the `DataSet` object is bound to the Flash Remoting custom server control using the `Flash.DataBind` method and returned to Flash.

Displaying a RecordSet in Flash with ActionScript

To display a `RecordSet` in a Flash UI component, you can use the `DataGlue ActionScript` file, which is installed with the Flash Remoting Components. You must first import the `ActionScript` file into your Flash application with the `include` directive, as the following example shows:

```
#include "DataGlue.as"
```

You can use the `DataGlue.bindFormatStrings` function to display the `RecordSet` in a Flash UI component, such as a `ComboBox` or a `Listbox`. The following example binds the result `RecordSet` to the `displayNames Listbox` UI component:

```
DataGlue.bindFormatStrings(displayNames, result, "#ContactName#", "#customerID#");
```

In the code, the last two arguments passed to the function (`#ContactName#` and `#customerID#`) are `RecordSet` column names. The `ContactName` column is displayed in the UI component, while the `customerID` column is returned when a user selects a particular record in the component.

The following `ActionScript` code connects to an `ASPX` page, returns a `RecordSet`, and displays the `RecordSet` in the `displayNames ComboBox` UI component:

```
//import the ActionScript classes
#include "NetServices.as"
#include "DataGlue.as"
//get a reference to the ASPX-based service
if (inited == null)
{
    inited = true;
    NetServices.setDefaultGatewayUrl("http://localhost/myASPApp/default.aspx");
    gatewayConnection = NetServices.createGatewayConnection();
    ASPXservice = gatewayConnection.getService("myASPApp", this);
    //call the ASPX page
    ASPXservice.myASPPage();
}
//handler for ASPX page results
function myASPPage_Result(result)
{
    DataGlue.bindFormatStrings(displayNames, result, "#ContactName#",
        "#customerID#");
}
```

You can also use the `DataGlue.BindFormatFunction` function to create custom formatting for your `RecordSets`. For more information on displaying `RecordSets` in `ActionScript`, Chapter 3, "Using Flash Remoting Data in ActionScript" on page 35.

Calling web services from Flash

Using the Flash Remoting web service adapter, you can call web services from Flash that are described by the Web Services Description Language (WSDL). You must first generate a local web service proxy to interact with web services. After you create the proxy, the ActionScript in your Flash application can then invoke web service methods through the proxy, which handles sending and receiving Simple Object Access Protocol (SOAP) messages with the remote web service.

In .NET, you can generate proxy assemblies with the WSDL Tool (`wSDL.exe`). Flash Remoting MX for .NET also uses the WSDL Tool to generate SOAP proxies for web services automatically from valid WSDL, either local or remote. In addition, Flash Remoting MX does not restrict you to .NET-based web services. Rather, any WSDL-described web service is available to Flash Remoting MX.

If you want to invoke web services using a .NET web service proxy assembly of your own that contains the web service definition, place the DLL file into the local assembly cache of your ASP.NET application. The proxy DLL must have the exact same name as the web service name, as described by the WSDL's service element. To invoke the web service proxy from ActionScript, supply the web service's fully qualified WSDL URL as the service address argument of the `gatewayConnection.getService` function, and use the web service's method names as the service function names.

Invoking web service methods using Flash Remoting MX

Flash Remoting MX uses the .NET WSDL Tool to generate the necessary proxy classes automatically by specifying a valid URL to a WSDL file or to a URL that can generate a WSDL file, such as a .NET ASMX file. To invoke a local web service in an ASMX file from Flash, you enter the URL to the file appended with `?wsdl`, as the following ActionScript example shows:

```
NetServices.setDefaultGatewayUrl("http://localhost/myASPApp/default.aspx");
gatewayConnection = NetServices.createGatewayConnection();
flashService = gatewayConnection.getService("http://localhost/myASPApp/
    ExampleWebService.asmx?wsdl", this);
```

In the `getService` function, you use the URL to the WSDL file, or a file capable of generating WSDL, as the service name. In the ASMX file, a `getMessage` method has been defined, as the following C# example shows:

```
[WebMethod]
public string getMessage()
{
    return "Flash Remoting makes web services easy!";
}
```

To call this method in ActionScript, you use the method name in the context of the `flashService` connection object, as the following ActionScript example shows:

```
flashService.GetMessage();
```

To display the results of the method invocation in Flash, you use an event handler, as the following example shows:

```
function getMessage_Result(result)
{
    serviceMessage.text = result;
}
function getMessage_Status(result)
{
    serviceMessage.text = error.description;
}
```

In the code, the results of the `getMessage` web service method call are displayed in the `serviceMessage` dynamic text field. For more information on handling results, see Chapter 2, “Handling service results” on page 27.

Invoking a remote web service from Flash

Using Flash Remoting MX for .NET, you can invoke any remote .NET-compatible web service directly from your Flash application with no .NET application development required. To find a remote web service, go to a public Universal Description, Discovery, and Integration (UDDI) registry, such as <http://www.xmethods.net>. Using the WSDL URL and method names found in the registry, you write ActionScript in your Flash application to invoke the web service.

To interact with remote web services, just like local web services, Flash Remoting MX uses the .NET framework’s WSDL Tool (`wSDL.exe`) to create web service proxies dynamically in the form of assemblies (*.dll). Remember, you must allow write and modify permissions for your ASP.NET application’s local assembly cache.

For example, the following ActionScript code connects to a Temperature web service (<http://www.xmethods.net/sd/2001/TemperatureService.wsdl>), which returns the local temperature by U.S. zip code:

```
#include "NetServices.as"
if (inited == null)
{
    inited = true;
    NetServices.setDefaultGatewayUrl("http://localhost/myASPApp/default.aspx");
    gatewayConnection = NetServices.createGatewayConnection();
    webService = gatewayConnection.getService("http://www.xmethods.net/sd/2001/
        TemperatureService.wsdl", this);
    webService.getTemp(zip.text);
}
function getTemp_Result(result)
{
    tempDisplay.text = result;
}
function getTemp_Status(result)
{
    tempDisplay.text = error.description;
}
```

In the ActionScript code, you replace the directory structure of the getService function argument with a URL that produces WSDL. The getTemp function maps to the web service method of the same name. The code assumes that zip represents an input text field, and tempDisplay represents a dynamic text field.

To see the web service proxy assembly (*.dll) that Flash Remoting MX creates, look in your local assembly cache for a DLL with the same name as the web service. For the Temperature web service, look for a DLL named TemperatureService.dll.

Calling ASP.NET assemblies from Flash

Using Flash Remoting MX, you can invoke .NET assembly files (*.dll) from Flash. In your ActionScript code, you use the fully qualified assembly or class file name in the `getService` function, and for the service function name, you use an assembly or class method name. On the server, you must place your DLL and class files in the local assembly cache.

Calling assemblies from Flash

In the class file, you reference the Flash Remoting assembly namespace `FlashGateway.IO` with the `using` directive, as the following C# example shows:

```
using System;
using FlashGateway.IO;
namespace FlashRemoting.EchoTests
{
    public class EchoClass
    {
        public EchoClass()
        {
            ///Public constructor... initialize any member fields here if need be.
        }
        public string echoString(string s)
        {
            return s;
        }
    }
}
```

In the ActionScript, you use the namespace and public class name defined in the class file, as the following example shows:

```
#include "NetServices.as"
#include "NetDebug.as"
if (inited == null)
{
    inited = true;
    NetServices.setDefaultGatewayUrl("http://localhost/myASPApp/default.aspx");
    gatewayConnection = NetServices.createGatewayConnection();
    classService =
        gatewayConnection.getService("FlashRemoting.EchoTests.EchoClass");
    classService.echoString(input.text);
}
function echoString_Result(result)
{
    stringDisplay.text = result;
}
function echoString_Status(result)
{
    stringDisplay.text = error.description;
}
```

In the code, you use the fully qualified class name (FlashRemoting.ClassService.EchoClass) in the `getService` function. To call an assembly method, you use the class method name (`echoString`) as defined in the class file.

Returning an ActionScript object from an assembly

You can use the `ASObject` class of the `FlashGateway.IO` namespace to create and populate ActionScript objects in ASP.NET and return the object to Flash. By passing ActionScript objects back and forth between the remote service and the Flash application, you can describe the data being passed with the `ASType` property of the `ASObject` class.

Creating an assembly that returns an ActionScript object

In the assembly, you create an instance of the `ASObject` class of the `FlashGateway.IO` namespace and return it to Flash. The `ASType` property lets you assign a name to the object for identification in Flash. To add values to the object, you use the `Add` method common to instances of the .NET Collections class, as the following C# example shows:

```
using System;
using FlashGateway.IO;
namespace FlashRemoting.ObjectTests
{
    public class ObjectClass
    {
        public ObjectClass()
        {
            ///Public constructor... initialize any member fields here if need be.
        }
        public ASObject returnObject()
        {
            ASObject aso = new ASObject();
            aso.ASType = "Calculator";
            aso.Add("x", 100);
            aso.Add("y", 300);
            Flash.Result = aso;
        }
    }
}
```

In the code, an instance of the `ASObject` object, named `aso`, is created, and the `ASType` property is used to identify the object as `Calculator`. The `Add` method inserts key-value pairs into the object. Finally, the `aso` object is returned to Flash using the `Flash.Result` variable.

Handling the ActionScript object in Flash

The following ActionScript handles the ActionScript object returned by the assembly:

```
#include "NetServices.as"
#include "NetDebug.as"

if (inited == null)
{
    inited = true;
    NetServices.setDefaultGatewayUrl("http://localhost/myASPApp/gateway.aspx");
    gatewayConnection = NetServices.createGatewayConnection();
    dataService =
        gatewayConnection.getService("FlashRemoting.ObjectTests.ObjectClass",
            this);
}

//Ask the server for some raw data...
dataService.returnObject();

//Provide a callback for getNumbers() when the data returns from the server
function returnObject_Result ( result )
{
    /*
    Because we expect the result to represent the state of a Calculator instance
    back from the server, we can assume the result will have an add() method.
    */
    resultBox.text = result.add();
}

/*
Rich Client Business Logic
*/
calc = function()
{
    this.x = 0;
    this.y = 0;
}
calc.prototype.subtract = function()
{
    return this.x - this.y;
}
calc.prototype.add = function()
{
    return this.x + this.y;
}
Object.registerClass("Calculator", calc);

stop();
```

Viewing Flash Remoting log entries

Flash Remoting MX writes error messages to the flash.log file in the local assembly cache. You can change the log file settings in the Windows Registry. The Flash Remoting registry keys are located at HKEY_LOCAL_MACHINE\SOFTWARE\Macromedia\Flash Remoting\1\Registration.

The following table lists the Flash Remoting registry keys:

Registry key	Description
Log File Size	<p>Maximum number of bytes that the flash.log file can reach before Flash Remoting MX stops writing entries. The default is 1 MB (1048576 bytes).</p> <p>The current log file used by Flash Remoting MX is named flash.log. When a log file reaches the file size limit, Flash Remoting saves the flash.log file as flash-<i>yyyymmhhmmss</i>.log. Then, Flash Remoting MX creates a new flash.log file and begins logging events to that file.</p>
Log Level	<p>Sets log message level, including Error (default), Warning, Information, Debug, or None.</p>

CHAPTER 8

Flash Remoting ActionScript Dictionary

Contents

- Overview of Flash Remoting ActionScript dictionary 140
- ActionScript element documentation conventions 141
- Contents of the dictionary 142
- DataGlue (object) 143
- NetConnection (object) 146
- NetDebug (object) 158
- NetServices (object) 160
- RecordSet (object) 164

Overview of Flash Remoting ActionScript dictionary

This dictionary describes the syntax and use of Macromedia Flash Remoting ActionScript elements in Macromedia Flash MX. To use examples in a script, copy the example code from the ActionScript Dictionary and paste it in the Actions panel in expert mode.

The dictionary lists all ActionScript elements, including operators, keywords, statements, actions, properties, functions, objects, components, and methods. For an overview of all dictionary entries, see “Contents of the dictionary” on page 142. The table in this section is a good starting point for looking up methods whose object or component class you do not know.

ActionScript follows the ECMA-262 Standard (the specification written by the European Computer Manufacturers Association) unless otherwise noted.

There are two types of entries in this dictionary:

- Object and component entries, which provide general information about built-in objects and the Flash MX components.
- Individual entries for operators, keywords, functions, variables, properties, methods, and statements.

Use the information in the sample entries to interpret the structure and conventions used in these two types of entries.

ActionScript element documentation conventions

The following table explains the conventions used for Flash Remoting-related ActionScript objects. Objects are listed alphabetically with all other elements in the dictionary.

Label	Description
Entry title	The entry title provides the name of the object or component. The object or component name is followed by a paragraph that contains general descriptive information.
Method and property table	Each object and component entry contains a table listing all of the associated methods. If the object or component has properties (often constants), These elements are summarized in an additional table. All of the methods and properties listed in these tables also have their own dictionary entries, which follow the object or component entry.
Constructor	If an object or component requires you to use a constructor to access its methods and properties, the constructor is described in each object or component entry. This description has all of the standard elements (syntax, description, and so on) of other dictionary entries.
Method and property listings	The methods and properties of an object or component are listed alphabetically after the object or component entry.

The following table explains the conventions used for all ActionScript methods and constructors:

Label	Description
Entry title	All entries are listed alphabetically. The alphabetization ignores capitalization, leading underscores, and so on.
Availability	This section tells which versions of the Flash Player support the element. This is not the same as the version of Flash used to author the content. For example, if you use the Flash MX authoring tool to create content for Flash Player 5, you can only use ActionScript elements that are available to Flash Player 5.
Usage	This section provides correct syntax for using the ActionScript element in your code. The required portion of the syntax is in code font, and the user provided code is in italicized code font. Brackets ([]) indicate optional parameters.
Parameters	This section describes any parameters listed in the syntax.
Return value	This section identifies what, if any, values the element returns.
Description	This section identifies the type of element (for example, an operator, method, function, and so on) and then describes how to use the element.
Example	This section provides a code sample demonstrating how to use the element.
See also	This section lists related ActionScript dictionary entries.

Contents of the dictionary

All dictionary entries are listed alphabetically. Methods that are associated with an object or component are listed with the object or component name. For example, the `replaceItemAt` method of the `RecordSet` object is listed as `RecordSet.replaceItemAt`. The following table lists all the ActionScript objects and methods for Flash Remoting:

DataGlue (object)

<code>DataGlue.bindFormatFunction</code>	<code>DataGlue.bindFormatStrings</code>
--	---

NetConnection (object)

Constructor for <code>NetConnection</code>	<code>NetConnection.getDebugID</code>
<code>NetConnection.addHeader</code>	<code>NetConnection.getService</code>
<code>NetConnection.call</code>	<code>NetConnection.setCredentials</code>
<code>NetConnection.close</code>	<code>NetConnection.setDebugID</code>
<code>NetConnection.connect</code>	<code>NetConnection.trace</code>
<code>NetConnection.getDebugConfig</code>	

NetDebug (object)

<code>NetDebug.trace</code>

NetServices (object)

<code>NetServices.createGatewayConnection</code>	<code>NetServices.setDefaultGatewayURL</code>
--	---

RecordSet (object)

Constructor for <code>RecordSet</code>	<code>RecordSet.isFullyPopulated</code>
<code>RecordSet.addItem</code>	<code>RecordSet.isLocal</code>
<code>RecordSet.addItemAt</code>	<code>RecordSet.removeAll</code>
<code>RecordSet.addView</code>	<code>RecordSet.removeItemAt</code>
<code>RecordSet.filter</code>	<code>RecordSet.replaceItemAt</code>
<code>RecordSet.getColumnNames</code>	<code>RecordSet.setDeliveryMode</code>
<code>RecordSet.getItemAt</code>	<code>RecordSet.setField</code>
<code>RecordSet.getItemID</code>	<code>RecordSet.sort</code>
<code>RecordSet.getLength</code>	<code>RecordSet.sortItemsBy</code>
<code>RecordSet.getNumberAvailable</code>	

DataGlue (object)

The `DataGlue` object is for Flash Remoting use only.

The `DataGlue` ActionScript methods let you bind `RecordSet` objects to Flash MX UI components. The `DataGlue` object offers a way to format data records for use in a `ListBox`, `ComboBox`, or other UI component.

To use `DataGlue`, you must first import the `DataGlue.as` file in the first frame of the Flash application with the `include` command, as follows:

```
#include "DataGlue.as"
```

Method summary for the `DataGlue` object

Method	Description
<code>DataGlue.bindFormatFunction</code>	Binds a data provider, such as a <code>RecordSet</code> object, to a data consumer, such as a <code>ListBox</code> component, and formats the data using a function, which you create.
<code>DataGlue.bindFormatStrings</code>	Binds a data provider, such as a <code>RecordSet</code> object, to a data consumer, such as a <code>ListBox</code> , and formats the provider data for the consumer.

DataGlue.bindFormatFunction

- Availability
- Flash Player 6.
 - Flash Remoting MX.

Usage `DataGlue.bindFormatFunction(dataConsumer, dataProvider, formatFunction)`

Parameters

Parameter	Description
<i>dataConsumer</i>	A <code>ListBox</code> or similar UI component to which you want to bind a data provider, such as a <code>RecordSet</code> object. The data consumer must support data provider objects (have a <code>setDataProvider</code> method).
<i>dataProvider</i>	A <code>RecordSet</code> object or other object that implements the standard data provider interface.
<i>formatFunction</i>	A user-defined function that takes a data record as a parameter. This function must return an <code>ActionScript</code> object that contains the fields <code>Label</code> and <code>Data</code> .

Return value **Nothing.**

Description **Method.** Binds a data provider, such as a `RecordSet` object, to a data consumer, such as a `ListBox` component, and formats the data using a function, which you create.

Example The following example binds the result `RecordSet` object to the `dataView2` UI component in the Flash application:

```
function myFormatFunction ( record )
{
    // the label is the parkname record field, translated to lower case
    var theLabel = record.parkname.toLowerCase();

    // the data is the length of the parkname record field
    var theData = record.parkname.length;

    // return the label and value to the caller
    return {label: theLabel, data: theData};
}
//call the bindFormatFunction method
DataGlue.bindFormatFunction(dataView2, result, myFormatFunction);
```

In this example, the `theLabel` variable is displayed in the UI component, and the `theData` variables are returned by the `getValue` function.

See also `DataGlue.bindFormatStrings`

DataGlue.bindFormatStrings

- Availability
- Flash Player 6.
 - Flash Remoting MX.

Usage `DataGlue.bindFormatStrings(dataConsumer, dataProvider, labelString, dataString)`

Parameters

Parameter	Description
<i>dataConsumer</i>	A Flash MX UI component, such as <code>ListBox</code> or <code>ComboBox</code> . The data consumer must support data provider objects (have a <code>setDataProvider</code> method).
<i>dataProvider</i>	A <code>RecordSet</code> or other object that implements the <code>dataProvider</code> interface
<i>labelString</i>	A format string that defines how to format fields of a data record as a label, which is the text that appears in the UI component. The format string is arbitrary text that can contain record field names enclosed in pound signs (#)
<i>dataString</i>	A format string that defines how to format fields of a data record as the data associated with the record

Return value **Nothing.**

Description **Method.** Binds a data provider, such as a `RecordSet` object, to a data consumer, such as a `ListBox`, and formats the data from the data provider for the consumer.

Example The following example binds the `myRecordSet` `RecordSet` object to the `myComboBox` UI component in the Flash application:

```
DataGlue.bindFormatStrings(myComboBox, myRecordSet, "#parkname# (#parktype#)",  
    "#city#, #state# #zipcode#");
```

In this example, the `parkname` and `parktype` variables are displayed in the UI component and the `city`, `state`, and `zipcode` variables are returned by the `getValue` function.

See also `DataGlue.bindFormatFunction`

NetConnection (object)

The `NetConnection` object manages a bidirectional connection between the Flash Player and the Flash Remoting service. The `NetServices` methods use the `NetConnection` object to call functions on and return results from application servers.

You normally create a `NetConnection` object by calling the `NetServices.createGatewayConnection` method. You can also create a `NetConnection` object directly as described in “Constructor for `NetConnection`” on page 147.

Method summary for the `NetConnection` object

Method	Description
<code>NetConnection.addHeader</code>	Adds a context header to the Action Message Format (AMF) packet structure.
<code>NetConnection.call</code>	Invokes a command or method on the server.
<code>NetConnection.close</code>	Closes the connection with the application server.
<code>NetConnection.connect</code>	Connects to the gateway on the application server.
<code>NetConnection.getDebugConfig</code>	Retrieves the <code>NetConnection</code> object's debug subscribed events <code>NetDebugConfig</code> object.
<code>NetConnection.getDebugID</code>	Retrieves the <code>NetConnection</code> object's debugging identifier.
<code>NetConnection.getService</code>	Creates a Flash Remoting service object.
<code>NetConnection.setCredentials</code>	Sends authorization credentials to Flash Remoting MX.
<code>NetConnection.setDebugID</code>	Sets a debug identifier for a <code>NetConnection</code> object.
<code>NetConnection.trace</code>	Sends a client trace message associated with the <code>NetConnection</code> to the <code>NetConnection Debugger</code> .

Constructor for NetConnection

- Availability
- Flash Player 6.
 - Flash Remoting MX.

Usage `new NetConnection()`

Parameters **None.**

Return value A `NetConnection` object.

Description **Constructor.** Creates an object that can connect the Flash Player to an application server.

Example **The following code creates a connection object and connects to a gateway.**

```
function onServerCon()  
{  
    //Makes a new connection object  
    gatewaycon = new NetConnection();  
  
    //Connects to the gateway on the server  
    gatewaycon.connect("http://www.mySite.com/flashservices/gateway");  
}
```

See also `NetConnection.connect`, `NetServices.createGatewayConnection`

NetConnection.addHeader

- Availability
- Flash Player 6.
 - Flash Remoting MX.

Usage `connectionObject.addHeader(name, mustUnderstand, object)`

Parameters

Parameter	Description
<i>name</i>	A string that is recognized by Flash Remoting MX and the application server that triggers some kind of processing.
<i>mustUnderstand</i>	A Boolean value that specifies whether the server must understand and process this header prior to handling any of the following headers or messages.
<i>object</i>	Any ActionScript object

Return value **Nothing.**

Description **Method.** Adds a context header to the Action Message Format (AMF) packet structure. This header is included with every AMF packet sent over this connection. This method is used by the `NetConnection.setCredentials` method; you do not normally use it directly in Flash applications.

NetConnection.call

- Availability
- Flash Player 6.
 - Flash Remoting MX.

Usage `connectionObject.call(remoteMethod, resultObject | null [, p1,...,pN])`

Parameters

Parameter	Description
<i>remoteMethod</i>	Parameter in the form <code>serviceName.methodName</code> , where the interpretation of <code>serviceName</code> depends on Flash Remoting MX and the application server.
<i>resultObject</i>	An object parameter that is needed only when the sender is expecting a result. The result object can be any user-defined object that implements a method named <code>onResult</code> . The object's <code>onResult</code> method will get called when the result arrives. If you do not need a result object, pass <code>null</code> .
<i>p1,...,pN</i>	Optional parameters to be passed to the specified method.

Return value **Nothing.**

Description **Method.** Invokes a command or method on the server. You must create a server-side function to execute the remote method.

Example The following `send` function checks for a message, sends it to a remote service, then clears the local string:

```
function send()
{
    if ( length(msg) > 0 ) {
        myConnection.call("messageService.message", null, msg);
    }
    msg = "";
}
```

NetConnection.close

Availability	<ul style="list-style-type: none">• Flash Player 6.• Flash Remoting MX.
Usage	<code>connectionObject.close()</code>
Parameters	None.
Return value	Nothing.
Description	Method. Makes the URL previously specified with <code>NetConnection.connect</code> into a null value, thereby removing the connection configuration for the remote gateway. Subsequent attempts to call Flash Remoting MX using the <code>NetConnection</code> object fail. After using the <code>NetConnection.close</code> method, you must call <code>NetConnection.connect</code> to define a new URL.
Example	The following <code>onServDiscon()</code> function closes the gatewaycon Flash Remoting connection: <pre>function onServDiscon() { gatewaycon.close(); }</pre>
See also	<code>NetConnection.connect</code>

NetConnection.connect

- Availability
- Flash Player 6.
 - Flash Remoting MX.

Usage `connectionObject.connect(targetURL)`

Parameters

Parameter	Description
<code>targetURL</code>	<p>The Flash Remoting URL</p> <p><i>[protocol://] host [:port] /appName</i></p> <p>The value of <code>appName</code> depends on the type of gateway and the particular configuration of Flash Remoting MX. For more information, see “Identifying the gateway” on page 21.</p> <p>If you omit <code>protocol://</code>, Flash assumes you want to connect to an application server using the nonsecure <code>http://</code> protocol. The other acceptable value for <code>protocol</code> is <code>https</code>.</p> <p>For example, the following URLs are formatted correctly:</p> <ul style="list-style-type: none">• <code>http://localhost:8500/flashservices/gateway</code>• <code>https://www.myCompany.com/myMainDirectory/securegateway.aspx/flashremoting/gateway</code>

Return value A Boolean value of `true` if the protocol part of the URL (the `http:` or `https:`) is valid, `false` otherwise.

Description Method. Defines the Flash Remoting URL that is used during Flash Remoting service function calls. This method does not communicate with the server. When Flash Remoting MX executes a service function call, it makes an HTTP or HTTPS connection to the application servers. This connection only persists until the results of the call are returned to the Flash application.

Example The following example configures the connection information for Flash Remoting MX to access the `mySite` server:

```
gatewaycon = new NetConnection();
gatewaycon.connect("http://www.mySite.com/flashservices/gateway");
```

See also `NetConnection.close`

NetConnection.getDebugConfig

Availability	<ul style="list-style-type: none">• Flash Player 6.• Flash Remoting MX.
Usage	<code>connectionObject.getDebugConfig()</code>
Parameters	None.
Return value	The NetConnection object's debug subscribed events NetDebugConfig object.
Description	Method. Retrieves the NetConnection object's debug subscribed events NetDebugConfig object.
Example	<p>The following example demonstrates the <code>getDebugConfig</code> method using the <code>NetServices</code> object:</p> <pre>#include "NetDebug.as" #include "NetServices.as" NetServices.setDefaultGatewayURL("http://www.mySite.com/flashservices"); NetServices.createGatewayConnection(); serviceObject = gatewayConnection.getService("myService", this); gatewayConnection.trace("I just created myService."); gatewayConnection.setDebugID("Gateway Connection"); var debugConfig = gatewayConnection.getDebugConfig();</pre>

NetConnection.getDebugID

- Availability
- Flash Player 6.
 - Flash Remoting MX.

Usage `connectionObject.getDebugID()`

Parameters **None.**

Return value **The NetConnection object's debug identifier.**

Description **Method. Retrieves the NetConnection object's debug identifier.**

Example **The following example demonstrates the `getDebugID` method using the `NetServices` object:**

```
#include "NetDebug.as"
#include "NetServices.as"
NetServices.setDefaultGatewayURL("http://www.mySite.com/flashservices");
NetServices.createGatewayConnection();
serviceObject = gatewayConnection.getService("myService", this);
gatewayConnection.trace("I just created myService.");
gatewayConnection.setDebugID("Gateway Connection");
var debugID = gatewayConnection.getDebugID();
```

See also `NetConnection.setDebugID`

NetConnection.getService

- Availability
- Flash Player 6
 - Flash Remoting MX.

Usage `connectionObject.getService(serviceName, defaultResponder)`

Parameters

Parameter	Description
<code>serviceName</code>	A string that identifies the Flash Remoting service name.
<code>defaultResponder</code>	The Flash movie object to receive the results of Flash Remoting service functions.

Return value A Flash Remoting service object.

Description **Method.** Creates a Flash Remoting service object, which allows access to application server functions.

Example The following example creates a connection to a gateway and creates a service object for accessing the myService application service. This example uses the `NetServices.createGateway` method, not the `NetConnection.connect` method, to create the connection to the gateway.

```
#include "NetServices.as"
NetServices.setDefaultGatewayURL("http://mySite.com/flashservices/gateway");
gatewayConnection = NetServices.createGatewayConnection();
serviceObject = gatewayConnection.getService("myService", this);
```

NetConnection.setCredentials

- Availability
- Flash Player 6.
 - Flash Remoting MX.

Usage `connectionObject.setCredentials(username, password)`

Parameters

Parameter	Description
<code>username</code>	Username to authenticate against in the application server.
<code>password</code>	Password to authenticate against in the application server.

Return value **Nothing.**

Description **Method.** Defines a set of credentials to be presented to the server on all subsequent requests. `setCredentials` can be called more than once. Currently, ColdFusion MX and JRun support this method.

Example The following example creates a gateway connection, sets the user credentials for the connection, and creates a service object to represent the remote service:

```
#include "NetServices.as"
NetServices.setDefaultGatewayURL("http://www.mySite.com/flasheservices/gateway");
gatewayConnection = NetServices.createGatewayConnection();
gatewayConnection.setCredentials(jdoe, 1234);
serviceObject = gatewayConnection.getService("myService", this);
```

NetConnection.setDebugID

- Availability
- Flash Player 6.
 - Flash Remoting MX.

Usage `connectionObject.setDebugID(id)`

Parameters

Parameter	Description
<code>id</code>	The new debug ID.

Return value **Nothing.**

Description **Method.** Sets a debug identifier for a `NetConnection` object. The debug identifier is passed as a top-level field of all debug events that are associated with this `NetConnection` object. A sequential number is generated as the debug identifier by default, but the type is not restricted.

Example The following example demonstrates the `setDebugID` method using the `NetServices` object:

```
#include "NetDebug.as"
#include "NetServices.as"
NetServices.setDefaultGatewayURL("http://mySite.com/flashservices");
NetServices.createGatewayConnection();
serviceObject = gatewayConnection.getService("myService", this);
gatewayConnection.trace("I just created myService.");
gatewayConnection.setDebugID("Gateway Connection");
```

See also `NetConnection.getDebugID`, `NetConnection.trace`

NetConnection.trace

- Availability
- Flash Player 6.
 - Flash Remoting MX.

Usage `connectionObject.trace(objectName)`

Parameters

Parameter	Description
<code>objectName</code>	Identifies any serializable ActionScript object.

Return value **Nothing.**

Description **Method.** Sends a client trace message associated with the `NetConnection` to the `NetConnection Debugger`. The trace message includes the connection's `Debug ID`.

Example **The following example demonstrates the `trace` method using the `NetServices` object:**

```
#include "NetDebug.as"
#include "NetServices.as"
NetServices.setDefaultGatewayURL("http://www.mySite.com/flasheservices");
NetServices.createGatewayConnection();
serviceObject = gatewayConnection.getService("myService", this);
gatewayConnection.trace("I just created myService.");
```

See also `NetConnection.setDebugID`, `NetDebug.trace`

NetDebug (object)

The `NetDebug` object lets developers trace function calls, parameters, and results among the Flash application, Flash Remoting, and the application server. You use the NetConnection Debugger (NCD) panel in the Flash MX authoring environment to view the debugging results.

To use `NetDebug`, you must first import the `NetDebug.as` file in the first frame of the Flash application with the `include` command, as follows:

```
#include "NetDebug.as"
```

Method summary for the `NetDebug` object

Method	Description
<code>NetDebug.trace</code>	Sends an <code>ActionScript</code> object to the <code>NetConnection</code> Debugger in a client trace event.

NetDebug.trace

- Availability
- Flash Player 6.
 - Flash Remoting.

Usage `NetDebug.trace(objectName)`

Parameters

Parameter	Description
<code>objectName</code>	Identifies the ActionScript object name.

Return value Sends an ActionScript object as a client trace event to the NetConnection Debugger

Description Method. Sends a serializable ActionScript object as a client trace event to the NetConnection Debugger. This trace event does not include connection information.

Example The following example sets a trace on the `serviceObject` object:

```
#include "NetServices.as"
#include "NetDebug.as"
NetServices.setDefaultGatewayURL("http://www.mySite.com/flashservices/gateway");
NetServices.createGatewayConnection();
serviceObject = gatewayConnection.getService("myService", this);
NetDebug.trace({level:"testing", message:"This is the message field of an object
               sent the NCD via trace."});
```

See also `NetConnection.trace`

NetServices (object)

The `NetServices` object is for Flash Remoting use only.

The `NetServices` object is a collection of methods that helps you create and use connections to services using Flash Remoting MX. The `NetServices` object helps you create and use `NetConnection` objects. For more information on the `NetConnection` object, see `NetConnection (object)`.

To use the `NetServices` object, you must first import the `NetServices.as` file in the first frame of the Flash application with the `include` command, as follows:

```
#include "NetServices.as"
```

Method summary for the `NetServices` object

Method	Description
<code>NetServices.createGatewayConnection</code>	Connects to Flash Remoting MX.
<code>NetServices.setDefaultGatewayURL</code>	Sets the default URL to connect to Flash Remoting MX.

NetServices.createGatewayConnection

- Availability
- Flash Player 6.
 - Flash Remoting MX.

Usage `NetServices.createGatewayConnection(URL)`

Parameters

Parameter	Description
<code>URL</code>	<p>A string that identifies a Flash Remoting service URL that will be used when running the movie in the Flash MX authoring environment.</p> <p><code>[protocol://] host [:port] /appName</code></p> <p>The value of <code>appName</code> depends on the type of gateway and the particular configuration of Flash Remoting MX. For more information, see "Identifying the gateway" on page 21.</p> <p>If you omit <code>protocol://</code>, Flash connects to Flash Remoting MX using the nonsecure <code>http://</code> protocol. Flash does not connect to the Flash Remoting service until a service function call. The other acceptable value for <code>protocol</code> is <code>https</code>.</p> <p>For example, the following URLs are formatted correctly:</p> <ul style="list-style-type: none">• <code>http://localhost:8500/flashservices/gateway</code>• <code>https://www.myCompany.com/myMainDirectory/secureServer</code>

Return value A `NetConnection` object that represents the gateway.

Description **Method.** Creates a `NetConnection` object, which is used to make connections to Flash Remoting MX on the application server. The Flash application only connects when you actually make a Flash Remoting service function call.

The URL parameter is optional. Flash Remoting MX determines the gateway URL as follows:

- 1 A URL specified in a `createGatewayConnection` method takes precedence over any other URL.
- 2 A URL specified in the deployed web page takes precedence over a default gateway URL. You specify the URL in the deployed web page by including it in the Flash player OBJECT tag, in the following format:

```
<OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
  codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/
  swflash.cab#version=6,0,0,0"
  WIDTH="100%"
  HEIGHT="100%"
  id="movieName">
  <PARAM NAME=flashvars VALUE="gatewayUrl=http://apps.mycompany.com/
  flashservices/gateway">
  <PARAM NAME=movie VALUE="movieName.swf">
  <PARAM NAME=quality VALUE=high>
  <PARAM NAME=bgcolor VALUE=#000099>
  <EMBED src="movieName.swf"
  FLASHVARS="gatewayURL=http://apps.mycompany.com/flashservices/gateway"
```

```
quality=high bgcolor=#000099
WIDTH="100%"
HEIGHT="100%"
NAME="movieName"
TYPE="application/x-shockwave-flash"
PLUGINSPAGE="http://www.macromedia.com/go/getflashplayer">
</EMBED>
</OBJECT>
```

In this example, the `flashvars` PARAM tag supplies the Flash Remoting service URL to Internet Explorer and the `FLASHVARS` attribute in `EMBED` tag specifies the URL to Netscape.

- 3 If you do not otherwise specify a gateway URL, Flash Remoting MX uses the gateway specified in a `NetServices.setDefaultGatewayURL` method.

Example The following code creates a `NetConnection` object for connections to the Flash Remoting gateway on `www.mySite.com`:

```
#include "NetServices.as"
NetServices.createGatewayConnection("http://www.mySite.com/flashservices/gateway");
```

See also `Constructor` for `NetConnection`, `NetServices.setDefaultGatewayURL`

NetServices.setDefaultGatewayURL

- Availability
- Flash Player 6.
 - Flash Remoting MX.

Usage `NetServices.setDefaultGatewayURL(URL)`

Parameters

Parameter	Description
<code>URL</code>	<p>A string that identifies a Flash Remoting service URL that will be used when running the movie in the Flash MX authoring environment.</p> <p><code>[protocol://] host [:port] /appName</code></p> <p>The value of <code>appName</code> depends on the type of gateway and the particular configuration of Flash Remoting MX. For more information, see “Identifying the gateway” on page 21.</p> <p>If you omit <code>protocol://</code>, Flash connects to Flash Remoting MX using the nonsecure <code>http://</code> protocol. Flash does not connect to the Flash Remoting service until a service function call. The other acceptable value for <code>protocol</code> is <code>https</code>.</p> <p>For example, the following URLs are formatted correctly:</p> <ul style="list-style-type: none">• <code>http://localhost:8500/flashservices/gateway</code>• <code>https://www.myCompany.com/myMainDirectory/secureServerApp/flashservices/gateway</code>

Return value **Nothing.**

Description **Method.** Sets the default Flash Remoting URL that Flash Remoting MX uses when it executes the `createGatewayConnection` method and you do not otherwise specify the gateway URL.

Note: If you specify `localhost` as the host in the `setDefaultGatewayURL` method, and you run your Flash application from outside the Flash development environment or stand-alone Flash player, Flash Remoting MX does not use `localhost` as the default gateway host. Instead, it replaces `localhost` and any port specified in the `setDefaultGatewayURL` method with the host and port specified to run the Flash application. For example, if you specify `http://localhost/flashservices/gateway` in the `setDefaultGatewayURL` method and start your Flash application by using the URL `http://apps.mycompany.com:8500/flashapps/myapp.swf` in a web page or browser, Flash Remoting MX uses `http://apps.mycompany.com:8500/flashservices/gateway` as the default gateway URL.

Example The following example establishes a default URL for Flash Remoting MX:

```
#include "NetServices.as"  
NetServices.setDefaultGatewayURL("http://www.mySite.com/flashservices/gateway");
```

See also `NetServices.createGatewayConnection`

RecordSet (object)

The `RecordSet` object is for Flash Remoting MX use only.

The `RecordSet` object lets you manipulate record sets returned from Flash Remoting or create client-side record sets. A record set is a list of records, with methods for fetching, accessing, and manipulating the list of records in various ways.

Record sets created on an application server usually consist of database query results.

Each record in a `RecordSet` object is represented by an untyped `ActionScript` object. In a `RecordSet` object, individual records are identified by an index number. The index starts at zero. When the record set is sorted or a record is added to or deleted from the record set, the index changes.

Each field of the record is represented by a field in the object. For a `RecordSet` object that originated from an application server, the field names are the same as the names of the fields as defined by the server-side record set. For local `RecordSet` objects, the field names are as defined in the original call to the `new RecordSet()` function.

To use the `RecordSet` object, you must use the `include` command to import the `NetServices` `ActionScript` class file, which also includes the `RecordSet` `ActionScript` class, in the first frame of a Flash application, as follows:

```
#include "NetServices.as"
```

Method summary for the RecordSet object

Method	Description
Constructor for <code>RecordSet</code>	Creates a new local <code>RecordSet</code> object.
<code>RecordSet.addItem</code>	Inserts a record into the <code>RecordSet</code> object.
<code>RecordSet.addItemAt</code>	Inserts a record into the <code>RecordSet</code> object at the specified index.
<code>RecordSet.addView</code>	Defines an object that will receive notifications when the <code>RecordSet</code> object changes.
<code>RecordSet.filter</code>	Creates a new <code>RecordSet</code> object that contains selected records from the original <code>RecordSet</code> object.
<code>RecordSet.getColumnNames</code>	Returns the names of all the columns of a <code>RecordSet</code> object.
<code>RecordSet.getItemAt</code>	Returns a record if the index is valid and the record is immediately available.
<code>RecordSet.getItemID</code>	Returns the record ID.
<code>RecordSet.getLength</code>	Returns the number of records in a <code>RecordSet</code> object.
<code>RecordSet.getNumberAvailable</code>	Returns the number of records that have been downloaded from the server.

Method	Description
<code>RecordSet.isFullyPopulated</code>	Determines whether a RecordSet object is fully populated or not.
<code>RecordSet.isLocal</code>	Determines whether a RecordSet object is available locally. Functionally equivalent to the <code>isFullyPopulated</code> method.
<code>RecordSet.removeAll</code>	Removes all records from the RecordSet object.
<code>RecordSet.removeItemAt</code>	Removes the specified record from the RecordSet object.
<code>RecordSet.replaceItemAt</code>	Replaces a record at the specified index.
<code>RecordSet.setDeliveryMode</code>	Changes the delivery mode of a record set from an application server.
<code>RecordSet.setField</code>	Replaces one field of a record with a new value.
<code>RecordSet.sort</code>	Sorts all the records by a user-specified compare function.
<code>RecordSet.sortItemsBy</code>	Sorts all records in the RecordSet object without making a new copy.

Constructor for RecordSet

- Availability
- Flash Player 6.
 - Flash Remoting.

Usage `new RecordSet (columnNames)`

Parameters

Parameter	Description
<code>columnNames</code>	An array of strings in which each string is a name of one of the record set columns.

Return value **New local** RecordSet object.

Description **Method. Creates a new local** RecordSet object. The RecordSet object is initially empty. **Local** RecordSet objects never download data from an application server; the data delivery-related RecordSet methods are not available to local RecordSet objects.

Example The following example creates a new local RecordSet object:

```
#include "NetServices.as"  
var productList = new RecordSet(["ProductName", "Price", "Color"]);
```

RecordSet.addItem

- Availability
- Flash Player 6.
 - Flash Remoting.

Usage `RecordSet.addItem(record)`

Parameters

Parameter	Description
<code>record</code>	The record to add.

Return value **Nothing.**

Description **Method.** Inserts a record into the `RecordSet` object. When you use the `addItem` method, avoid the following conditions:

- The record parameter is not an object.
- The record parameter has unknown or missing fields.
- The `RecordSet` object is associated with an application server and not fully populated yet.

Error handling

Error condition	What happens	Error message
The record parameter is not an object.	The record is added to the <code>RecordSet</code> object.	
The record parameter has unknown or missing fields.	The record is added to the <code>RecordSet</code> object.	
The <code>RecordSet</code> object is server-associated but not fully populated.	No record is added to the <code>RecordSet</code> , and an error message is reported to the Flash MX output window and Debug Console.	Operation not allowed on partial <code>RecordSet</code> objects.

Example **The following example demonstrates the `addItem` method:**

```
#include "NetServices.as"
var productList = new RecordSet(["ProductName", "Price", "Color"]);
var itemToAdd = {ProductName: "magicbubbles"; Price: 1, Color: 0x987654};
productList.addItem(itemToAdd);
```

See also [Constructor for RecordSet](#), [RecordSet.addItemAt](#)

RecordSet.addItemAt

- Availability
- Flash Player 6.
 - Flash Remoting.

Usage `RecordSet.addItemAt(index, record)`

Parameters

Parameter	Description
<i>index</i>	The index number of the record.
<i>record</i>	The record to add.

Return value **Nothing.**

Description **Method.** Inserts a record into the `RecordSet` object at the specified index. When you use the `addItemAt` method, avoid the following conditions:

- Index out of range.
- The record parameter is not an object.
- The record parameter has unknown or missing fields.
- The `RecordSet` object is associated with an application server and not fully populated yet.

Error handling

Error condition	What happens	Error message
Index is less than zero.	The <code>RecordSet</code> object is not changed.	
Index is greater than the length of the <code>RecordSet</code> object.	The <code>RecordSet</code> object length is extended, and the record is added to the <code>RecordSet</code> object.	
The record parameter is not an object.	The record is added to the <code>RecordSet</code> object.	
The record parameter contains unknown or missing fields.	The record is added to the <code>RecordSet</code> object.	
The <code>RecordSet</code> object is associated with an application server and not fully-populated yet.	No change is made to the <code>RecordSet</code> object, and an error message is reported to the Flash MX Output Window and Debug Console.	Operation not allowed on partial <code>RecordSet</code> objects.

Example **The following example demonstrates the `addItemAt` method:**

```
#include "NetServices.as"
var productList =new RecordSet(["ProductName","Price","Color"]);
var itemToAdd = {ProductName:"magicbubbles", Price:1, Color:"0x987654"};
productList.addItemAt(0,itemToAdd);
```

See also `Constructor for RecordSet`, `RecordSet.addItem`

RecordSet.addView

- Availability
- Flash Player 6.
 - Flash Remoting.

Usage `RecordSet.addView(object)`

Parameters

Parameter	Description
<i>object</i>	An ActionScript object to be notified when the RecordSet object changes.

Return value **Nothing.**

Description **Method.** Defines an ActionScript object that will receive notifications when the RecordSet object changes. The object must contain a `modelChanged` function, which takes one parameter, and an event descriptor object. The following table describes the event descriptor messages:

Message	Description
{event:"addRows", firstRow:xxx, lastRow:yyy}	Row numbers xxx through yyy have been added.
{event:"allRows"}	All records have arrived from the server. In other words, theRecordSet object is now fully populated.
{event:"deleteRows", firstRow:xxx, lastRow:yyy}	Row numbers xxx through yyy have been deleted.
{event:"fetchrows", firstRow:xxx, lastRow:yyy}	Row numbers xxx through yyy have been requested from the server, but have not arrived yet.
{event:"sort"}	The record set has been sorted.
{event:"updateAll"}	The RecordSet object has changed in some way, such as a new view being added.
{event:"updateRows", firstRow:xxx, lastRow:yyy}	Row numbers xxx through yyy have changed in some way.

Example The following example shows the use of the `addView` method. The `modelChanged` function displays a trace message with the event type each time the record set changes.

```
#include "NetServices.as"

function modelChanged(info)
{
    trace(info.event);
}

var productList = new RecordSet(["Name","Price","Color"]);
```

```
// whenever productList changes, call this.modelChanged().
productList.addView(this);

// modify the RecordSet object, and see if "modelChanged" gets called
productList.addItem({Name: "milk", Price: 3.50, Color: "0xffffffff"});
productList.addItem({Name: "eggs", Price: 1.75, Color: "0xffffffff"});
```

See also [Constructor for RecordSet](#)

RecordSet.filter

- Availability
- Flash Player 6.
 - Flash Remoting.

Usage `RecordSet.filter(filterFunction, context)`

Parameters

Parameter	Description
<i>filterFunction</i>	An ActionScript function that takes one or two parameters and returns <code>true</code> or <code>false</code> . The first parameter is a single record from the <code>RecordSet</code> object. The second, optional, parameter is a context value that the function uses to determine whether to include the record in the result. The function must return <code>true</code> if the record should be included in the result <code>RecordSet</code> object.
<i>context</i>	A context value supplied by the caller. This value is the second parameter to the filter function.

Return value A new `RecordSet` object that contains a reference, not a copy, to all the records that were selected by the *filterFunction* function.

Description **Method.** Creates a new `RecordSet` object by calling the *filterFunction* function once for each record in the `RecordSet` object and collecting all records, for which the *filterFunction* function returns `true`, into a new `RecordSet` object. The order of the records in the new `RecordSet` object is the same as their order in the original `RecordSet` object. The order in which the original `RecordSet` object is traversed during the filtering process is not defined.

If used on a `RecordSet` object that is not fully-populated, only the currently available records are filtered. The new `RecordSet` object does not inherit the original `RecordSet` object's list of views, nor any association with a server-side `RecordSet` object.

Example The following example demonstrates how to build a `ListBox` UI component that shows a filtered view of a `RecordSet` object:

```
#include "NetServices.as"
var allFlights =new RecordSet(["flight","numberOfStops"]);
allFlights.addItem({flight: "123", numberOfStops: 0});
allFlights.addItem({flight: "321", numberOfStops: 3});
allFlights.addItem({flight: "456", numberOfStops: 1});
allFlights.addItem({flight: "654", numberOfStops: 2});
function flightFilter(aRecord, requestedNumberOfStops)
{
    return (aRecord.numberOfStops <= requestedNumberOfStops);
}
myListBox.setDataProvider(allFlights.filter(flightFilter, 2));
```

See also `RecordSet.addView`, `RecordSet.sort`

RecordSet.getColumnNames

- Availability
- Flash Player 6.
 - Flash Remoting.

Usage `RecordSetObject.getColumnNames()`

Parameters **None.**

Return value **An array of strings.** The array is either the same array that you passed into the `RecordSet` constructor, or the equivalent array for an application server-associated `RecordSet` object.

Description **Method.** Returns the names of all the columns of a `RecordSet` object as an array of strings.

Example **The following example demonstrates the `getColumnNames` method:**

```
#include "NetServices.as"
var productList = new RecordSet(["ProductName", "Price", "Color"]);
var titles = productList.getColumnNames();
_root.firstColumnTitle.text = titles[0];
_root.secondColumnTitle.text = titles[1];
```

See also `Constructor for RecordSet`

RecordSet.getItemAt

- Availability
- Flash Player 6.
 - Flash Remoting.

Usage `RecordSet.getItemAt(index)`

Parameters

Parameter	Description
<i>index</i>	The record number to retrieve.

Return value A record.

Description Method. Returns a record if the index is valid and the record is immediately available. If the requested record index number is less than zero or greater than the largest record index number, it returns null. If the index number is valid but the requested record has not yet been downloaded, it returns the string "in progress". Remember that `RecordSet` object indexes change when they are sorted or records are deleted or added.

Error handling

Error condition	What happens	Error message
Index out of range	Null is returned, and an error message is reported to the Flash MX output window and Debug Console.	RecordSet warning 104: getItemAt(index) index out of range

Example The following example demonstrates the `getItemAt` method:

```
#include "NetServices.as"
var productList = new RecordSet(["ProductName","Price","Color"]);
productList.addItem({ProductName : "Spoon", Price :77, Color : "0x987654"});
var record = productList.getItemAt(0);
```

See also Constructor for `RecordSet`, `RecordSet.addItemAt`

RecordSet.getItemID

- Availability
- Flash Player 6.
 - Flash Remoting.

Usage `RecordSet.getItemID(index)`

Parameters

Parameter	Description
<i>index</i>	The index number of the record.

Return value A unique identification (ID) corresponding to the record, at the specified index. Returns null if the index is out of range.

Description **Method.** Returns a unique ID corresponding to the record, at the specified index. The `RecordSet` object assigns each record a unique ID. The ID is not part of the record; it is a separate item that is associated with the record internally within the `RecordSet` object. Unlike a record index, its ID will not change when the `RecordSet` object is sorted or when records are added or deleted. When a record is deleted, its ID is retired and will never be used again in this `RecordSet` object. Also, the ID is used by the `ListBox` object to maintain the end user's selection when the `RecordSet` object changes.

Example The following example retrieves a record ID from a `RecordSet` object:

```
#include "NetServices.as"
var productList =new RecordSet(["ProductName","Price","Color"]);
productList.addItem({ProductName : "Spoon",Price :77,Color : "0x987654"});
var recordID =productList.getItemID(0);
```

RecordSet.getLength

- Availability
- Flash Player 6.
 - Flash Remoting.

Usage `RecordSetObject.getLength()`

Parameters **None.**

Return value **The number of records in a RecordSet object.**

Description **Method. Returns the number of records in a RecordSet object.**

Example **The following example demonstrates the `getLength` method:**

```
#include "NetServices.as"
var productList =new RecordSet(["ProductName","Price","Color"]);
productList.addItem({ProductName : "Spoon",Price :77,Color :0x987654});
productList.addItem({ProductName : "Niblick",Price :33,Color :660000});
trace("There are " + productList.getLength() + " products in the department.");
```

RecordSet.getNumberAvailable

- Availability
- Flash Player 6.
 - Flash Remoting.

Usage `RecordSet.getNumberAvailable()`

Parameters **None.**

Return value **An integer between zero and the total size of the record set, inclusive.**

Description **Method.** Returns the number of records that have been downloaded from the server. The count does not include records that have been requested but not yet arrived. For local `RecordSet` objects, this will always return the same value as the `getLength` method.

Example **The following example function uses the `getNumberAvailable` method:**

```
#include "NetServices.as"
function reportDownloadStatus(aRecordSet)
{
    trace(aRecordSet.getNumberAvailable()+ "of " + aRecordSet.getLength() + "records
        have been downloaded.");
}
```

See also `RecordSet.getLength`, `RecordSet.isFullyPopulated`, `RecordSet.isLocal`

RecordSet.isFullyPopulated

- Availability
- Flash Player 6.
 - Flash Remoting.

Usage `RecordSet.isFullyPopulated()`

Parameters **None.**

Return value **Returns true if the RecordSet object is fully populated or false if the RecordSet object is not fully populated.**

Description **Method. Determines whether a RecordSet object is fully populated, that is, has all its records. Local RecordSet objects are always fully populated. RecordSet objects provided by application servers are fully populated after all of their records have been downloaded from the application server.**

A RecordSet object must be fully populated before you can use any of the following data editing and manipulation methods:

- addItem
- addItemAt
- replaceItemAt
- setField
- removeItem
- removeAll
- filter
- sort

This method is functionally identical to the `RecordSet.isLocal` method.

Example The following example demonstrates the `isFullyPopulated` method using a local RecordSet object:

```
#include "NetServices.as"
var productList =new RecordSet(["ProductName","Price","Color"]);
var itemToAdd = {ProductName:"magicbubbles",Price:1,Color:"0x987654"};
var itemToAdd2 = {ProductName:"kiddyshampoo",Price:7,Color:"0x876543"};
productList.addItem(itemToAdd);
productList.addItem(itemToAdd2);
if (productList.isFullyPopulated())
{
    editButton.enable();
}
```

See also `RecordSet.isLocal`, `RecordSet.getLength`

RecordSet.isLocal

- Availability
- Flash Player 6.
 - Flash Remoting.

Usage `RecordSet.isLocal()`

Parameters **None.**

Return value **Returns true if the RecordSet object is local and false if records remain to be retrieved from the application server.**

Description **Method. Determines whether a RecordSet object is local or associated with an application server, and records remain to be retrieved from the application server. This method is functionally identical to the RecordSet.isFullyPopulated method.**

Example **The following example demonstrates the isLocal method:**

```
#include "NetServices.as"
var productList =new RecordSet(["ProductName","Price","Color"]);
if (productList.isLocal())
{
    _root.tellUser.text ="This RecordSet object is not associated with a server.";
}
```

See also `RecordSet.isFullyPopulated`

RecordSet.removeAll

- Availability
- Flash Player 6.
 - Flash Remoting.

Usage `RecordSet.removeAll()`

Parameters **None.**

Return value **Nothing.**

Description **Method.** Removes all records from the record set. Do not use the `removeAll` method, when the `RecordSet` object is associated with an application server and not fully populated yet.

Error handling

Error condition	What happens	Error message
The <code>RecordSet</code> object is server-associated and not fully populated yet.	No change is made to the <code>RecordSet</code> object, and an error message is reported to the Flash MX output window and Debug Console.	Operation not allowed on partial <code>RecordSet</code> objects.

Example The following example demonstrates the `removeAll` method:

```
#include "NetServices.as"
var productList =new RecordSet(["ProductName","Price","Color"]);
var itemToAdd ={ProductName:"magicbubbles",Price:1,Color:"0x987654"};
productList.addItem(itemToAdd);
productList.removeAll();
```

See also [Constructor](#) for `RecordSet`, `RecordSet.removeItemAt`

RecordSet.removeItemAt

- Availability
- Flash Player 6.
 - Flash Remoting.

Usage `RecordSet.removeItemAt(index)`

Parameters

Parameter	Description
<i>index</i>	The index number of the record.

Return value **Nothing.**

Description **Method.** Removes the specified record. The associated record ID is never used again in the `RecordSet` object. When you use the `removeItemAt` method, avoid the following conditions:

- Index out of range.
- The `RecordSet` object is associated with an application server and not fully populated yet.

Error handling

Error condition	What happens	Error message
The <code>RecordSet</code> object is server-associated but not fully populated.	No change is made to the <code>RecordSet</code> object, and an error message is reported to the Flash MX output window and Debug Console.	Operation not allowed on partial <code>RecordSet</code> objects.
Index out of range	The <code>RecordSet</code> object is not changed.	

Example **The following example demonstrates the `removeItemAt` method:**

```
#include "NetServices.as"
var productList = new RecordSet(["ProductName", "Price", "Color"]);
var itemToAdd = {ProductName:"magicbubbles", Price:1, Color:"0x987654"};
productList.addItem(itemToAdd);
productList.removeItemAt(0);
```

See also [Constructor for RecordSet](#), [RecordSet.getItemID](#), [RecordSet.addItem](#)

RecordSet.replaceItemAt

- Availability
- Flash Player 6.
 - Flash Remoting.

Usage `RecordSet.replaceItemAt(index, record)`

Parameters

Parameter	Description
<i>index</i>	The index number of the record.
<i>record</i>	The record to add.

Return value **Nothing.**

Description **Method.** Replaces a record in the `RecordSet` object at the specified index. The specified index must identify an existing record. The record's contents are replaced with the contents of the record parameter. The record's ID does not change. When you use the `replaceItemAt` method, avoid the following conditions:

- Index out of range.
- The record parameter is not an object.
- The record parameter has unknown or missing fields.
- The `RecordSet` object is associated with an application server and not fully populated yet.

Error handling

Error condition	What happens	Error message
The <code>RecordSet</code> object is server-associated but not fully populated.	No change is made to the <code>RecordSet</code> object, and an error message is reported to the Flash MX output window and Debug Console.	Operation not allowed on partial <code>RecordSet</code> objects.
Index out of range	The <code>RecordSet</code> object is not changed.	
Unknown field name	The <code>RecordSet</code> object is not changed.	

Example **The following example demonstrates the `replaceItemAt` method:**

```
#include "NetServices.as"
var productList = new RecordSet(["ProductName", "Price", "Color"]);
var itemToAdd = {ProductName:"magicbubbles", Price:1, Color:"0xb00fed"};
productList.addItem(itemToAdd);
myListBox.setDataProvider(productList);
var replacementItem = {ProductName:"kiddyShampoo", Price:2, Color:"0xd00d1e"};
productList.replaceItemAt(0, replacementItem);
```

See also [Constructor for RecordSet](#), [RecordSet.addItem](#), [RecordSet.addItemAt](#)

RecordSet.setDeliveryMode

- Availability
- Flash Player 6.
 - Flash Remoting.

Usage `RecordSet.setDeliveryMode(mode, pagesize, numPrefetchPages)`

Parameters

Parameter	Description
<i>mode</i>	Identifies the delivery mode. The options are <code>ondemand</code> (default), <code>fetchall</code> , and <code>page</code> .
<i>pagesize</i>	(Optional) In <code>page</code> mode, what the page size is in <code>fetchall</code> mode, how many records to fetch in each server request. The default is 25.
<i>numPrefetchPages</i>	(Optional) In <code>page</code> mode, the number of pages to prefetch. The default is 0, fetch only the required page.

Return value **Nothing.**

Description Method. Changes the delivery mode of a record set associated with an application server. At any time, a `RecordSet` object associated with an application server is operating in a particular data-delivery mode. The new mode setting takes effect immediately, except that pending application server requests are allowed to complete. You can change mode settings and delivery mode parameters.

Until you call this method for a `RecordSet` object, it operates in `ondemand` mode. When using `fetchall` mode, you can supply a `pagesize` parameter. The entire record set will be fetched from the application server in a series of requests, and each request will fetch only the number of records specified in the `pagesize` parameter.

When using `page` mode, you can supply the `pagesize` and `preFetchPages` parameters. In `page` mode, when you request a record using the `getItemAt` method, the `RecordSet` object ensures that `numPrefetch` pages after the page containing the requested record are either already available in the client or requested from the server. If `numPrefetch` pages is zero, only the current page containing the requested record is fetched.

When the `RecordSet` object is fully populated, the `setDeliveryMode` method has no effect.

Error handling

Error condition	What happens	Error message
Unknown mode string	No change is made to the <code>RecordSet</code> object, and an error message is reported to the Output Window and Debug Console.	<code>SETDELIVERYMODE: unknown mode string</code>

Example The following examples demonstrate the `setDeliveryMode` method:

```
myRecordSet.setDeliveryMode("fetchall");  
myRecordSet.setDeliveryMode("page", 25, 2);
```

See also `RecordSet.getItemAt`, `RecordSet.isFullyPopulated`

RecordSet.setField

- Availability
- Flash Player 6.
 - Flash Remoting.

Usage `RecordSet.setField(index, fieldName, value)`

Parameters

Parameter	Description
<i>index</i>	The index number of the record.
<i>fieldName</i>	The field name to replace.
<i>value</i>	The value to insert into the field.

Return value Nothing.

- Description
- Method. Replaces one field of a record with a new value. When you use the `setField` method, avoid the following conditions:
- Index out of range.
 - The `RecordSet` object is associated with an application server and not fully populated yet.
 - Unknown field name. The *fieldName* parameter does not equal a valid column name.

Error handling

Error condition	What happens	Error message
The <code>RecordSet</code> object is server-associated but not fully populated.	No change is made to the <code>RecordSet</code> object, and an error message is reported to the Flash MX output window and Debug Console.	Operation not allowed on partial <code>RecordSet</code> objects.
Index out of range	The <code>RecordSet</code> object is not changed.	
Unknown field name	The <code>RecordSet</code> object is not changed.	

Example The following example demonstrates the `setField` method:

```
#include "NetServices.as"
var productList =new RecordSet(["ProductName","Price","Color"]);
var itemToAdd ={ProductName:"magicbubbles",Price:1,Color:"0x987654"};
productList.addItem(itemToAdd);
var replacementValue ="flashmx";
productList.setField(0,"ProductName",replacementValue);
```

See also Constructor for `RecordSet`, `RecordSet.replaceItemAt`

RecordSet.sort

- Availability
- Flash Player 6.
 - Flash Remoting.

Usage `RecordSet.sort(compareFunction)`

Parameters

Parameter	Description
<i>compareFunction</i>	An ActionScript comparison function that determines the sorting order. Given the arguments A and B, the comparison function returns a value as follows: <ul style="list-style-type: none">• -1 if A appears before B in the sorted sequence.• 0 if A = B.• 1 if A appears after B in the sorted sequence.

Return value Nothing.

Description Method. Sorts all the records using a user-specified compare function. The `sort` method sorts all the records in place, without making a new copy. The order is determined by the user-supplied *compareFunction* function. The original order is not remembered.

The `sort` method is approximately 10 times slower than the `sortItemsBy` method.

All registered views receive the following callback:

```
View.modelChanged("sort");
```

When you use the `sort` method, avoid sorting a `RecordSet` object that is associated with an application server and not fully populated yet.

Error handling

Error condition	What happens	Error message
The <code>RecordSet</code> object is not fully populated.	No change is made to the <code>RecordSet</code> object, and an error message is reported to the Flash MX output window and Debug Console.	Operation not allowed on partial <code>RecordSet</code> objects.

Example The following example demonstrates how to sort an `employeeList` `RecordSet` object by multiple keys:

```
function orderBySeveralKeys(aRecord, bRecord)
{
    var aKey = aRecord.location + aRecord.department + aRecord.lastName;
    var bKey = bRecord.location + bRecord.department + bRecord.lastName;
    if (aKey < bKey)
    {
        return -1;
    }
    else if (aKey > bKey)
    {
        return 1;
    }
}
```

```
    }  
    else  
    {  
        return 0;  
    }  
}  
employeeList.sort(orderBySeveralKeys);
```

See also `RecordSet.addView`, `RecordSet.filter`, `RecordSet.sortItemsBy`

RecordSet.sortItemsBy

- Availability
- Flash Player 6.
 - Flash Remoting.

Usage `RecordSet.sortItemsBy(fieldName, direction)`

Parameters

Parameter	Description
<i>fieldName</i>	The name record field that is the sort key.
<i>direction</i>	(Optional) Specifies the sort direction. A value of "DESC" specifies descending sorting. Any other value is interpreted as ascending sorting, the default.

Return value Returns `true` if the sort succeeds, or `false` if any errors occurred.

Description Method. Sorts all records in the `RecordSet` object without making a new copy. The sort key value for each record is the contents of the field identified by the field ID. The original order is not saved.

Sorting large `RecordSet` objects can take a long time. `RecordSet` objects that contain fewer than 2000 records should take less than one second on a computer with a Pentium 3 processor. Sort times increase rapidly as the number of records grows.

When two or more records have the same sort key value, they are not sorted in a special way. The original order of the records might not be preserved.

If the `fieldName` parameter identifies a field that does not exist in one or more records, then `null` is used as the key value for those records. `Null` sorts lower than any other value.

The `sortItemsBy` method may only be used on fully populated `RecordSet` objects.

All registered views will receive the following callback:

```
View.modelChanged("sort", 1, lastRecordNumber);
```

Error handling

Error condition	What happens	Error message
The <code>RecordSet</code> object is server-associated but not fully populated.	No change is made to the <code>RecordSet</code> object, and an error message is reported to the Flash MX output window and Debug Console.	Operation not allowed on partial <code>RecordSet</code> objects.

Example The following example demonstrates the `sortItemsBy` method:

```
#include "NetServices.as"
var productList = new RecordSet(["ProductName", "Price", "Color"]);
var itemToAdd = {ProductName:"magicbubbles", Price:1, Color:"0x987654"};
var itemToAdd2 = {ProductName:"kiddyshampoo", Price:7, Color:"0x876543"};
var itemToAdd3 = {ProductName:"dishes", Price:2, Color:"0x098763"};
productList.addItem(itemToAdd);
productList.addItem(itemToAdd2);
productList.addItem(itemToAdd3);
```

```
productList.sortItemsBy("ProductName");  
productList.sortItemsBy("Price","DESC");
```

See also `RecordSet.addView`, `RecordSet.filter`, `RecordSet.sort`

A

- Action Message Format *See* AMF
- ActionScript
 - connection object, creating 19
 - data types, converting from
 - application server 39
 - data types, converting to
 - application server 37
 - debugging 66
 - dictionary overview 140
 - function results, handling in 111
 - service object, creating 22
 - typed objects, using 43
 - XML object, sending to Java 114
- ActionScript arrays
 - accessing in ColdFusion
 - components 79
 - accessing in ColdFusion pages 72
- ActionScript collections
 - accessing in ColdFusion
 - components 79
 - accessing in ColdFusion pages 72
- ActionScript data types
 - converting from ColdFusion 41
 - converting to ColdFusion 71
- ActionScript objects
 - accessing in ColdFusion
 - components 80
 - accessing in ColdFusion pages 73
 - creating an assembly 135
 - element documentation
 - conventions 141
 - handling in Flash 136
 - returning from an assembly 135
- adapters, service 3
- AddHeader client event message 63
- addHeader NetConnection object
 - method 148
- addItem RecordSet object
 - method 167
- addItemAt RecordSet object
 - method 168
- addView RecordSet object
 - method 169
- ADO.NET objects
 - using with Flash Remoting
 - MX 128
- AMF
 - debugging messages 62
 - understanding 3
- amf app_server event type 62
- amfheaders app_server event
 - type 62
- AmfMethodCall app_server event
 - message 64
- AmfRequestHeader app_server event
 - message 64
- AmfResponseCall app_server event
 - message 65
- AmfResponseHeader app_server
 - event message 64
- AmfStatusCall app_server event
 - message 65
- app_server
 - event type 61
 - NetConnection Debugger event
 - messages 64
- application
 - building a sample 9
 - building Flash with Flash
 - Remoting MX 5
 - building the Flash 10
 - structure 14
- application server
 - authenticating to 23
 - converting data types to
 - ActionScript 39
 - converting data types to, from
 - ActionScript 37
 - data conversion 40
 - debugger events 61
 - tool 5
- Arguments scope 81
- arguments, using named with
 - ColdFusion 25
- ASP.NET
 - application 121
 - assemblies 134
 - pages 122
 - state management 125
- ASPX page
 - getting a reference to 122
 - invoking in ActionScript 123
 - using the Flash Remoting custom
 - server control in 123
- assemblies
 - ASP.NET, calling 134
 - creating 135
 - returning an ActionScript object
 - from 135
- authentication
 - ColdFusion MX 24
 - JRun 4 24
 - to the application server 23

B

- bindFormatFunction DataGlue
 - object method 56, 144
- bindFormatStrings DataGlue object
 - method 56, 145
- Boolean data in ColdFusion 41

C

C#

- converting data to
 - ActionScript 39
- converting data types from
 - ActionScript 37
- Call client event message 64
- call NetConnection object
 - method 149
- CF.http function 82
- CF.query function 85
- cfargument tag 78
- cfcatch tag 90
- cflogin tag 88
- cftry tag 90
- classes, Flash Remoting 16, 17
- client
 - event type 61
 - NetConnection Debugger event messages 63
- Close client event message 64
- close NetConnection object
 - method 150
- code-behind files
 - using the Flash Remoting namespace in 124
- ColdFusion
 - ActionScript data conversion
 - issues 41
 - Boolean data in 41
 - converting data to
 - ActionScript 39
 - converting data types from
 - ActionScript 37
 - data types, converting to
 - ActionScript 71
 - debugging 62
 - functions with named arguments,
 - calling 25
 - NetConnection Debugger event messages 65
 - numeric data 41
- coldfusion app_server event type 62
- ColdFusion components
 - ActionScript collections,
 - accessing 79
 - ActionScript objects,
 - accessing 80
 - Arguments scope 81
 - cfargument tag 78
 - component metadata 81

- error handling 90
- Flash Remoting MX and 77
- Flash service name 77
- pageable record sets 77
- parameter passing 78
- record sets, increments 77
- record sets, returning 77
- returning results from 77
- security 88

ColdFusion MX

- authenticating users 24
- delivering RecordSet data
 - from 53
- error handling 90
- Flash Remoting MX and 69
- Flash variable scope 71
- incremental record sets 76, 78
- server-side ActionScript 82
- using Flash Remoting MX with
 - ColdFusion components 77
 - using Flash Remoting MX with ColdFusion pages 70
 - web services 86

ColdFusion pages

- ActionScript object access 73
- arrays and 72
- error handling 90
- Flash Remoting MX and 70
- Flash scope 71
- Flash service name 70
- pageable record sets 76
- parameter passing 71
- record sets, increments 76
- record sets, returning 75
- returning results from 74
- security 88
- structures and 72

ColdFusion security

- cflogin tag 88
- example 88
- roles 89
- with Flash Remoting MX 88

- components. *See* Flash MX UI components
- configuring Flash Remoting MX 19
- Connect client event message 63
- connect NetConnection object
 - method 151

- connecting
 - creating the connection object 19
 - creating the service object 22
 - defining the event handlers 25
- connections, debugging 66
- constructor
 - NetConnection ActionScript object 147
 - RecordSet ActionScript object 166
- controller, model-view-controller pattern 6
- createGatewayConnection
 - NetServices object
 - method 161
 - specifying gateway in 20
- credentials, resetting 24
- custom server control
 - using in ASPX pages 123

D

- data conversion
 - application server 40
 - ColdFusion to ActionScript 41
- data types
 - and Flash Remoting MX 36
 - converting from ActionScript to application server 37
 - converting from application server to ActionScript 39
- DataGlue ActionScript object
 - bindFormatFunction
 - method 144
 - bindFormatStrings method 145
- class file, including 18
- described 18
- method summary 143
- methods, using 56
- reference overview 143

- debugger. *See* NetConnection Debugger
- debugging
 - NetDebug.trace method 66
 - See also* NetConnection Debugger specific connections 66
- design patterns
 - .NET 118
 - applying to Flash Remoting MX 6

- facade 7
 - using other with Flash Remoting MX 7
- development environment understanding 5
- E**
- EJB methods
 - invoking in ActionScript 100
- EJBHome object
 - getting a reference to 100
- EJBs
 - connection considerations 23
 - Flash application example 101, 111
 - using with Flash Remoting MX 100
- environment, understanding development 5
- Error app_server event message 64
- error app_server event type 62
- error handling
 - error object 31
 - example 30, 32
 - hierarchy 31
 - in ColdFusion 90
 - Status event 31
 - strategies 32
- error object, handling 31
- event types, debugger 61
- events
 - application server 64
 - client 63
 - ColdFusion 65
 - common debugger information 62
- F**
- facade design pattern 7
- filter RecordSet object method 171
- Flash
 - application, building 5, 10
 - ASP.NET pages, calling from 122
 - custom tag, properties 120
 - web services, calling from 131
 - XML object, returning to 114
- Flash Communication Server debugging 62
 - NetConnection Debugger events 65
- Flash MX authoring environment 5
- Flash MX UI components
 - using with RecordSet objects 54
- Flash Remoting
 - ActionScript dictionary overview 140
 - application structure 14
 - architecture 3
 - classes 16, 17
 - custom server control, using in ASPX pages 123
 - development environment 5
 - namespace, using in code-behind files 124
 - remote services, building the Flash application 10
 - sample application, building 9
 - service adapters 3
- Flash Remoting components
 - ActionScript classes 16
 - authenticating Flash movies 23
 - calling service functions 25
 - creating the connection object 19
 - creating the service object 22
 - including ActionScript classes 18
 - overview 14
 - passing parameters 25
- Flash Remoting MX
 - .NET framework 118
 - about 1, 2
 - ADO.NET objects, using with 128
 - and data types 36
 - ASP.NET state management, using with 125
 - building Flash applications with 5
 - ColdFusion components and ColdFusion MX and 69
 - ColdFusion pages and 70
 - ColdFusion results, accessing 74
 - configuring 19
 - design patterns, applying 6
 - design patterns, using other with 7
 - EJBs, using with 100
 - for .NET, overview 118
 - for Java, overview 92
 - Java classes and JavaBeans, using with 94
- JMX, using with 107
- JRun security with 113
- model-view-controller pattern, using with 6
- server-side ActionScript, using with 82, 109
- servlets and JSPs, using with 104
- web services, ColdFusion 86
- XML and Java 114
- Flash scope 71
- Flash service name
 - ColdFusion components 77
 - ColdFusion pages 70
- Flash.Pagesize 71
- Flash.Params 71
 - ActionScript objects and 73
 - parameter referencing 71
 - using 71
- Flash.Result 71
- flashcomm_server NetConnection
 - Debugger event types 62
- function results, handling in ActionScript 111
- functions, specifying 26
- G**
- gateway
 - specifying in a web page 21
 - specifying in createGatewayConnection NetServices object method 20
- gateway URL
 - determining 19
 - identifying 21
- getColumnNames RecordSet object method 172
- getDebugConfig
 - NetConnection object method 152
 - NetDebugConfig object method 67
- getDebugID NetConnection object method 153
- getItemAt RecordSet object method 173
- getItemID RecordSet object method 174
- getLength RecordSet object method 175

getNumberAvailable RecordSet
object method 176
getService NetConnection object
method 154
specifying a responder object 28

H

handling
error hierarchy 31
error strategies 32
errors 31
errors with ColdFusion 90
function results in
ActionScript 111
result hierarchy 27
result strategies 27
service results 27
Hello World application
building with Flash Remoting
MX 9
http
client event type 61
debugging 61
httpheaders app_server event
type 62
HttpRequestHeader app_server event
message 64

I

incremental record sets 76
Information app_server event
message 65
isFullyPopulated RecordSet object
method 177
isLocal RecordSet object
method 178

J

Java
converting data to
ActionScript 39
converting data types from
ActionScript 37
Flash Remoting MX for 92
results 111
serializable objects 45
using XML with Flash Remoting
MX 114
Java application servers
EJBs, using with Flash Remoting
MX 100

Java classes and JavaBeans, using
with Flash Remoting
MX 94
JMX, using with Flash Remoting
MX 107
logging 115
returning complex objects 111
servlets and JSPs, using with Flash
Remoting MX 104
Java classes, using with Flash
Remoting MX 94
JavaBeans, using with Flash Remoting
MX 94
JMX, using with Flash Remoting
MX 107
JRun
JMX 107
using security with Flash Remoting
MX 113
JRun 4
authenticating users 24

L

log entries, viewing 115
logging Java application servers 115
logging out users 24

M

MBeans
calling JMX from Flash 107
getting a reference to 107
invoking methods in
ActionScript 108
Microsoft .NET
Flash Remoting design
patterns 118
Flash Remoting directory
structure 120
overview 118
model, model-view-controller
pattern 6
model-view-controller pattern, using
with Flash Remoting MX 6
MVC *See* model-view-controller

N

named arguments, using with
ColdFusion 25
namespace, using in code-behind
files 124

NetConnection ActionScript object
addHeader method 148
call method 149
close method 150
connect method 151
constructor 147
described 17
getDebugConfig method 152
getDebugID method 153
getService method 22, 154
method summary 146
reference overview 146
setCredentials method 23, 155
setDebugID method 66, 156
trace method 66, 157
NetConnection Debugger
ActionScript for 66
application server event
messages 64
client event messages 63
ColdFusion event messages 65
common event information 62
configuring output in
ActionScript 66
event types 61
Flash Communication Server
events 65
using 60
NetDebug ActionScript object
class file, including 18
described 17
method summary 158
reference overview 158
trace method 66, 159
NetDebugConfig ActionScript
object 66
getting with getDebugConfig
method 67
NetServices ActionScript object
class file, including 18
createGatewayConnection
method 19, 161
described 17
method summary 160
reference overview 160
setDefaultGatewayURL
method 163
notifications, using with RecordSet
objects 50
numeric data in ColdFusion 41

- O**
- objects, using in Flash Remoting applications 43
- P**
- pageable record sets 53, 76
- R**
- realtime_server, NetConnection
 - Debugger event types 62
- record sets
 - about 46
 - ColdFusion components, returning from 77
 - ColdFusion pages, returning from 75
 - debugging 61
 - delivering data from ColdFusion MX 53
 - filtering 52
 - incremental 76, 78
 - pageable 53, 76
 - sorting 51
 - See also* RecordSet ActionScript object
- RecordSet
 - displaying in Flash with ActionScript 130
- recordset
 - app_server event type 62
 - client event type 61
- RecordSet ActionScript object
 - addItem method 167
 - addItemAt method 49, 168
 - addView method 50, 169
 - bindFormatFunction method 56
 - bindFormatStrings method 56
 - changing data in 49
 - constructor 166
 - creating 48
 - data, editing 53
 - described 18
 - event descriptor messages 51
 - filter method 171
 - filtering 52
 - Flash MX UI components 54
 - getColumnNames method 172
 - getItemAt method 173
 - getItemID method 174
 - getLength method 175
 - getNumberAvailable method 176
 - getting data values, getting 48
 - incremental record sets 76, 78
 - information, getting 48
 - isFullyPopulated method 177
 - isLocal method 178
 - items, adding 49
 - method summary 164
 - methods 47
 - methods and properties, using 48
 - notifications, using 50
 - pageable 53
 - record fields, renaming 50
 - records, adding 49
 - records, removing 49
 - records, replacing 50
 - records, replacing and renaming 50
 - reference overview 164
 - removeAll method 179
 - removeItemAt method 49, 180
 - replaceItemAt method 50, 181
 - See also* record sets
 - setDeliveryMode method 182
 - setField method 183
 - sort method 184
 - sorting 51
 - sortItemsBy method 51, 186
 - using 46
 - using directly in components 55
 - values, returning 48
- reference
 - ASPX page, getting 122
 - EJBHome, getting 100
 - MBean, getting 107
 - server-side ActionScript, getting 109
- remote functions
 - specifying 26
- remote service
 - Flash application, building 10
 - remote web service, invoking from Flash 132
- removeAll RecordSet object
 - method 179
- removeItemAt RecordSet object
 - method 180
- replaceItemAt RecordSet object
 - method 181
- responder object
 - specifying 25
 - specifying in service functions 29
 - specifying with getService method 28
- Result client event message 63
- results
 - complex Java objects, returning 111
 - handling 27
 - handling example 29
 - handling hierarchy 27
 - Java 111
 - returning from RecordSet objects 48
 - strategies for handling 27
- roles, web services and 89
- rtmp client event type 61
- S**
- security
 - Flash Remoting MX with JRun 113
 - implementing 23
 - logging out users 24
 - with ColdFusion 88
- serializable Java objects 45
- server-side ActionScript
 - CF.http function 82
 - CF.query function 85
 - functions, invoking 109
 - getting a reference to 109
 - overview 82
 - using with Flash Remoting MX 109
- service adapters 3
- service functions
 - calling 25
 - defining the event handlers 25
 - example 30
 - handling results 27
 - passing parameters 25
 - result handling example 29
 - result object 29
 - result-handling hierarchy 27
 - result-handling strategies 27
 - specifying 26
 - specifying a responder object in 29

- service object
 - creating 22
 - EJB considerations 23
- service, specifying 22
- session variables
 - getting and setting with Flash Remoting MX 126
- setCredentials function 88
- setCredentials NetConnection object
 - method 155
- setDebugID NetConnection object
 - method 66, 156
- setDefaultGatewayURL NetServices
 - object method 163
 - URL resolution 20
 - using 20
- setDeliveryMode RecordSet object
 - method 182
- setField RecordSet object
 - method 183
- SOAP
 - converting data to
 - ActionScript 39
 - converting data types from
 - ActionScript 37
- sort RecordSet object method 184
- sortItemsBy RecordSet object
 - method 186
- state management
 - using ASP.NET with Flash Remoting MX 125
- Status client event message 63

T

- trace
 - client event type 61
 - Flash Communication Server event
 - type 62
 - NetConnection object
 - method 66, 157
 - NetDebug object method 66, 159
- Trace client event message 63
- typed objects
 - using ActionScript 43

U

- URL, determining the gateway 19
- users, logging out 24

V

- view, model-view-controller
 - pattern 6
- Visual Basic
 - converting data to
 - ActionScript 39
 - converting data types from
 - ActionScript 37

W

- web page, specifying gateway in 21
- web service methods
 - calling using ColdFusion 86
 - invoking using Flash Remoting MX 131
- web services
 - calling from Flash 131
 - invoking remote from Flash 132
 - invoking with ColdFusion 86
 - security, roles for 89
 - using ColdFusion 86
- WSDL file 86

X

- XML
 - object, returning to Flash 114
 - object, sending to Java 114
 - using in Flash Remoting
 - applications 57
 - using with Flash and Java 114