Using Server-Side ActionScript in ColdFusion MX

Part Number ZCF60M600

Acknowledgments
Project Management: Stephen M. Gilson

Writing: Stephen B. Gilson

Editing: Linda Adler, Noreen Maher

First Edition: May 2002

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

CONTENTS

ABOUT THIS BOOK

Using Server-Side ActionScript in ColdFusion MX describes how to access ColdFusion services using ActionScript, the scripting language associated with Macromedia Flash.

Contents

Developer resources

Macromedia, Inc. is committed to setting the standard for customer support in developer education, 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 ColdFusion	Detailed product information for ColdFusion and related topics	http://www.macromedia.com/coldfusion
Macromedia ColdFusion Support Center	Professional support programs that Macromedia offers	http://www.macromedia.com/support/coldfusion
ColdFusion Online Forums	Access to experienced ColdFusion developers through participation in the Online Forums, where you can post messages and read replies on many subjects relating to ColdFusion	http://webforums.macromedia.com/coldfusion/
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 ColdFusion development, including online discussion groups, Knowledge Base, technical papers, and more	http://www.macromedia.com/desdev/developer/
Reference Desk	Development tips, articles, documentation, and white papers	http://www.macromedia.com/v1/developer/TechnologyReference/index.cfm
Macromedia Alliance	Connection with the growing network of solution providers, application developers, resellers, and hosting services creating solutions with ColdFusion	http://www.macromedia.com/partners/

About ColdFusion documentation

ColdFusion documentation is designed to provide support for the complete spectrum of participants. The print and online versions are organized to let you quickly locate the information that you need. The ColdFusion online documentation is provided in HTML and Adobe Acrobat formats.

Printed and online documentation set

The ColdFusion documentation set consists of the following titles:

Book	Description
Installing ColdFusion MX	Describes system installation and basic configuration for Windows NT, Windows 2000, Solaris, Linux, and HP-UX.
Administering ColdFusion MX	Describes how to connect your data sources to the ColdFusion Server, configure security for your applications, and how to use ClusterCATS to manage scalability, clustering, and load-balancing for your site.
Developing ColdFusion MX Applications with CFML	Describes how to develop your dynamic web applications, including retrieving and updating your data, and using structures and forms.
Getting Started Building ColdFusion MX Applications	Contains an overview of ColdFusion features and application development procedures. Includes a tutorial that guides you through the process of developing a sample ColdFusion application.
Using Server-Side ActionScript in ColdFusion MX	Describes how Macromedia Flash movies executing on a client browser can call ActionScript code running on the ColdFusion Server. Includes examples of server-side ActionScript and a syntax guide for developing ActionScript pages on the server.
Migrating ColdFusion 5 Applications	Describes how to migrate a ColdFusion 5 application to ColdFusion MX. This book describes the code compatibility analyzer that evaluates your ColdFusion 5 code to determine any incompatibilities within it.
CFML Reference	Provides descriptions, syntax, usage, and code examples for all ColdFusion tags, functions, and variables.
CFML Quick Reference	A brief guide that shows the syntax of ColdFusion tags, functions, and variables.
Working with Verity Tools	Describes Verity search tools and utilities that you can use for configuring the Verity K2 Server search engine, as well as creating, managing, and troubleshooting Verity collections.

Viewing online documentation

All ColdFusion documentation is available online in HTML and Adobe Acrobat Portable Document Format (PDF) files. To view the HTML documentation, open the following URL on the web server running ColdFusion: http://*web_root*/cfdocs/dochome.htm.

ColdFusion documentation in Acrobat format is available on the ColdFusion product CD-ROM.

Getting answers

One of the best ways to solve particular programming problems is to tap into the vast expertise of the ColdFusion developer communities on the ColdFusion Forums. Other developers on the forum can help you figure out how to do just about anything with ColdFusion. 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. The Forums is a great resource for learning ColdFusion, but it is also a great place to see the ColdFusion developer community in action.

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
Technical support	Macromedia offers a range of telephone and web-based support options. Go to http://www.macromedia.com/support/coldfusionfor a complete description of technical support services. You can make postings to the ColdFusion Support Forum (http://webforums.macromedia.com/coldfusion) at any time.
Sales	Toll Free: 888.939.2545 Tel: 617.219.2100 Fax: 617.219.2101 E-mail: sales@macromedia.com Web: http://www.macromedia.com/store

CHAPTER 1
Introduction

Macromedia ColdFusion Server MX includes the Macromedia Flash Remoting service, a module that lets Macromedia Flash developers create server-side ActionScript. These ActionScripts can directly access ColdFusion query and HTTP features through two new ActionScript functions: `CF.query` and `CF.http`. This chapter introduces the concepts and basic requirements for creating server-side ActionScript for ColdFusion MX.

Contents

About server-side ActionScript

Macromedia ColdFusion MX includes a module called the Macromedia Flash Remoting service that acts as a broker for interactions between Macromedia Flash MX and ColdFusion MX. In addition to support for a range of object types, Flash Remoting also lets you reference an ActionScript file that lives on a ColdFusion server.

The ability to create server-side ActionScript provides a familiar way for Flash developers to access ColdFusion resources without having to learn CFML (ColdFusion Markup Language). One benefit of this feature is that it lets you partition data-intensive operations on the server, while limiting the amount of network transactions necessary to get data from server to client.

In addition, this feature lets you logically separate the Flash presentation elements of your applications from the business logic. Now you have the option of creating ActionScript files that reside on the server to partition this processing away from your client applications.

You have a very simple interface for building queries using server-side ActionScript, and an equally simple interface for invoking these queries from your client-side ActionScript.

Client-side ActionScript requirements

On the client side, you only need a small piece of code that establishes a connection to the Flash Remoting service and references the server-side ActionScript you want to use.

For example (note embedded comments):

```
// This #include is needed to connect to the Flash Remoting service
#include "NetServices.as"

// This line determines where Flash MX should look for the Flash Remoting service.
// Ordinarily, you enter the URL to your ColdFusion server.
// Port 8100 is the Flash Remoting service default.
NetServices.setDefaultGatewayUrl("http://mycfserver:8100");

// With the Flash Remoting service URL defined, now you can create a connection.
gatewayConnnection = NetServices.createGatewayConnection();

// Here's where you reference the server-side ActionScript.
// In this case the stockquotes script file lives in the web root of the
// ColdFusion server identified previously. If it lived in a subdirectory
// of the web root called "mydir," you would reference it
// as "mydir.stockquotes".
stockService = gatewayConnnection.getService("stockquotes", this);

// This line invokes the getQuotes() method defined in the stockquotes
// server-side ActionScript.
stockService.getQuotes("macr");
```

```
// Once the record set is returned, you handle the results.
// This part is up to you.
function getQuotes_Result ( result )
{
  // Do something with results
}
```

Note: The two new server-side ActionScript functions, `CF.query` and `CF.http`, are not supported in client-side ActionScript.

Server-side requirements

The option of creating ActionScript that executes on the server helps leverage your knowledge of ActionScript, while providing direct access to ColdFusion query and HTTP features. The `CF.query` and `CF.http` ActionScript functions let you perform ColdFusion HTTP and query operations:.

Note: On the server side, ActionScript files use the extension .asr.

For example, the following server-side ActionScript builds on the client-side code shown previously:

```
// Filename: stockquotes.asr
// Here is the getQuotes method invoked in the client-side ActionScript.
// It accepts a single stock quote symbol argument.
function getQuotes(symbol)
{
  // Query some provider for the specified stock quote and return the
  // results. In this case, the getQuotesFromProvider method is
  // defined elsewhere in this ActionScript.
  data = getQuotesFromProvider(symbol);
  // Now return the data to the client.
  // Note that this example does not include any of the error checking
  // logic you would normally use prior to returning the data.
  return data;
}
```

The `getQuotes` function conducts the stock quote request and returns the results of the request to the client as a RecordSet object.

Software requirements

To use server-side ActionScripts, you must have the following software installed:

- Macromedia Flash MX
- Macromedia ColdFusion MX
- Flash Remoting Components

For more information about these products, go to www.macromedia.com.

More about server-side ActionScript

The ability to create server-side ActionScript provides a familiar way for Flash developers to access ColdFusion resources without having to learn CFML.

Through the Flash Remoting service, Flash developers can leverage their knowledge of ActionScript to access ColdFusion query and HTTP features. The only new things you need to learn to make use of this feature are the use of two new ActionScript functions that let you perform ColdFusion HTTP and query operations, and a few lines of setup code on the client side.

This is a feature that lets you create data-intensive Flash-based applications. Creating ActionScript files that reside on the server helps the business logic part of your application from the user interface.

Location of server-side ActionScript files

You can place ActionScript files (`*.asr`) on the server anywhere below the web server's root directory. To specify subdirectories of the web root or a virtual directory, use package dot notation. For example, in the following assignment code, the stockquotes.asr file is located in the mydir/stock/ directory:

```
stockService = gatewayConnnection.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 subdirectories of that mapping.

Benefits

Server-side ActionScript lets your ActionScript engineers use their knowledge of ActionScript to write code for the backend of their Flash applications, which can mean more meaningful levels of interactivity for your users. Your Flash applications can share a library of server-side ActionScript functions, which means you can define functions that are specifically tailored to your own business.

You could, for example, create a server-side ActionScript that defines a whole library of SQL query methods. With these query methods defined on the server-side, your Flash designers only have to invoke the specific query function they want to return data to their Flash MX movies. They do not have to write any SQL, and they do not have to create a new query every time they need to retrieve data from a ColdFusion data source. It is a way of creating reusable queries that your entire Flash design team can use.

Coding the ColdFusion query and HTTP operations in ActionScript is very straightforward. The `CF.query` and `CF.http` functions provide a well-defined interface for building SQL queries and HTTP operations that is based on the legendary simplicity of ColdFusion.

For example, the following is a typical server-side ActionScript function definition that returns query data:

```
// This function shows a basic CF.query operation using only
// arguments for data source name and for SQL.
function basicQuery()
{
  mydata = CF.query({datasource:"customers",
    sql:"SELECT * FROM myTable"});
     return mydata;
}
```

What to do next

If you are already familiar with ActionScript, you only need to know a few things to get started:

- How to establish a connection with the Flash Remoting service using client-side ActionScript. For details, see "Connecting to the Flash Remoting service" on page 8
- How to reference server-side ActionScripts and methods. For details, refer to "Calling server-side ActionScript functions" on page 9
- How to code the server-side `CF.query` and `CF.http` functions. For details, see Chapter 5, "ColdFusion ActionScript Functions" on page 29.

The following table shows where to find more information about using ActionScript with ColdFusion MX:

For more information about	See
ActionScript	Macromedia Flash MX documentation, and www.macromedia.com
Coding client-side and servier-side ActionScript	Chapter 5, "ColdFusion ActionScript Functions" on page 29
Writing query functions using server-side ActionScript	Chapter 3, "Using the CF.query Function" on page 13
Writing HTTp functions using server-side ActionScript	Chapter 4, "Using the CF.http Function" on page 23

CHAPTER 2
Connecting to the Flash Remoting Service

This chapter describes how to establish a connection to the Macromedia Flash Remoting service, and includes other programming details about referencing server-side ActionScript.

Contents

Connecting to the Flash Remoting service

Before you can use functions defined in your server-side ActionScript, you must first perform several initial set up tasks.

First, you connect the Macromedia Flash movie to the server-side Flash Remoting service. The following steps create a Flash Remoting service connection:

1 Include `NetServices.as` in your Flash movie.

2 Specify the default Flash Remoting service URL and create a connection to the service.

See the following sections for more details:

- "Including the necessary ActionScript classes" on page 8
- "Establishing a connection to the Flash Remoting service" on page 8

After you perform this setup, you can use the functions defined in your server-side ActionScript.

Including the necessary ActionScript classes

You must include a series of ActionScript classes in the first frame of the Flash movie that will be using server-side ActionScript functions. The following command includes the `NetServices` class:

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

The following command includes the `NetDebug` class:

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

The `NetServices` include statement is required for server-side ActionScript. The `NetDebug` include statement is optional.

Establishing a connection to the Flash Remoting service

The Flash Remoting service serves as a broker for calls to server-side ActionScripts.

To use server-side ActionScript:

1 Identify the Flash Remoting service URL as an argument in the `NetServices.setDefaultGatewayUrl` function; for example:

```
NetServices.setDefaultGatewayURL("http://localhost:8100/flashservices")
```

You must specify a ColdFusion MX Server hostname. The default port number for the Flash Remoting service is 8100.

2 Create the gateway connection using the `NetServices.createGatewayConnection` function; for example:

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

Calling server-side ActionScript functions

After you connect to the Flash Remoting service, you call functions that are defined in your server-side ActionScript, and return results.

To call a functions:

1 Create an instance of the server-side ActionScript using the `getService` function. This function instantiates the server-side ActionScript as an object to be used on the client side.

2 Call a function defined in your server-side ActionScript object.

3 Handle the function results in ActionScript.

For more information, see the following sections:

- "Creating an instance of the server-side ActionScript" on page 9
- "Calling a server-side ActionScript function" on page 9
- "Using the function results in ActionScript" on page 10

Creating an instance of the server-side ActionScript

To access the server-side ActionScript, first you must create an instance of it so it can be invoked in your client-side ActionScript. To create an instance of your server-side ActionScript, use the `gatewayConnection.getService` function; for example:

```
albumService = gatewayConnection.getService("recordsettest", this)
```

In the example, `recordsettest` represents the name of the server-side ActionScript file, without the file extension `.asr`.

Calling a server-side ActionScript function

To use the functions in a server-side ActionScript, you use dot notation to specify the object name name followed by the function name; for example:

```
albumService.getAlbum("The Color And The Shape", "1999");
```

In the example, `albumService` is the instance of the server-side ActionScript and `getAlbum` is a function that passes two arguments, `"The Color and The Shape"` and `"1999"`.

Note: Arguments must occur in the order defined in the function declaration.

Using the function results in ActionScript

To use the results returned by server-side ActionScript, you must create a corresponding **results function**, which is a function defined specifically to handle the results returned by a function that calls a server-side ActionScript. The results function uses a special naming convention that ties it to the function that calls the server-side ActionScript. For example, if you defined a client-side ActionScript function called `basicCustomerQuery`, you also must create a results function called `basicCustomerQuery_result`.

The results returned by server-side ActionScript functions differ somewhat depending on whether you are using `CF.http` or `CF.query`.

The `CF.query` function returns a record set, which you manipulate using methods available in the RecordSet ActionScript class object.

The `CF.http` function returns simple text strings through properties that you reference in your server-side ActionScript.

For more information, see the following sections:

- "Using results returned by the CF.query function" on page 10
- "Using results returned by the CF.http function" on page 11

Using results returned by the CF.query function

You access the data returned in a `CF.query` record set using functions in the RecordSet ActionScript object. Functions in the RecordSet object let you determine how many records are in the record set, what the column names are, and so on. RecordSet functions also let you pull the query data out of the record set. To do so, you reference a specific row number in the record set and use the `getItemAt` RecordSet function, as in the following example:

```
// This function populates a Flash text box with data in the first row
// of the record set under the "email" column name.
function selectData_result ( result )
{
   stringOutput.text =  result.getItemAt(0)["email"];
   _root.employeesView.setDataProvider(result);
}
```

In the example, the column name is referenced in the `getItemAt` function between square brackets []. The rows returned by the `CF.query` function are counted starting from one, so the first row is row 0 (zero), the second row is row 1, and so on.

For more information about coding the server-side `CF.query` function, see Chapter 3, "Using the CF.query Function" on page 13.

Using results returned by the CF.http function

The `CF.http` server-side ActionScript function returns data as simple text. You write server-side functions that reference the properties available in the object returned by the `CF.http` function. These properties store the file content of the retrieved file, HTTP status codes, the MIME type of the returned file, and so on. On the client side, you create return functions to handle data returned by the `CF.http` function. You write these functions to handle simple text data.

For more information, see Chapter 4, "Using the CF.http Function" on page 23.

Global and request scope objects

Global and request scope objects are implicitly available in all server-side ActionScript. The following table describes these scope objects:

Scope name	Type	Description
config	Global	Initialization information for the server-side ActionScript adapter. Class: `javax.servlet.ServletConfig`
application	Global	The context for the current web application. The context defines methods that provide, for example, the MIME type of a file that can be used to write to a log file. There is one context per web application. Class: `javax.servlet.ServletContext`
request	Request	An object containing client request information. The object provides data including parameter name and values, attributes, and an input stream. Class: `HttpServletRequest` (subtype of `javax.servlet.ServletRequest`)
response	Request	An object to assist in sending a response to the client. It provides HTTP-specific functionality in sending a response. Do not use the OutputStream or PrintWriter to send data back to the client. Class: `HttpServletResponse` (subtype of `javax.servlet.ServletResponse`)

For more information about these scope objects, see the documentation on the javax.servlet class, found at http://java.sun.com.

CHAPTER 3
Using the CF.query Function

This chapter describes how to retrieve data from a database using the `CF.query` ActionScript function.

Contents

Overview

The `CF.query` function lets you populate Macromedia Flash MX movie elements with data retrieved from a ColdFusion data source.

To pull data into your Flash MX movie from a ColdFusion data source:

1 Create a server-side ActionScript file that performs queries against a ColdFusion data source.

2 Write ActionScript code in your Flash MX movie that references your ActionScript file (.asr) on the ColdFusion server.

You create a server-side ActionScript to execute the query and returns the data in a record set to the client—your Flash MX movie. You can use methods in the RecordSet ActionScript object on the client to manipulate data in the record set and present data in your Flash MX movie.

Note: Client-side ActionScript files use the .as extension. Server-side ActionScript files use the .asr extension, for "ActionScript remote."

Publishing dynamic data

The server-side ActionScript feature in ColdFusion Server MX lets you write server-side ActionScript files to perform queries against ColdFusion data sources. You must first understand the following:

- How to code database queries in the server-side ActionScript file using the `CF.query` ActionScript function. See "CF.query," in Chapter 5.
- How to reference the server-side ActionScript file in your Flash MX movie. See Chapter 2, "Connecting to the Flash Remoting Service" on page 7.

Using the `CF.query` function, you can do the following tasks:

- Create user login interfaces that validate users against a ColdFusion data source.
- Populate form elements and data grids with data from a ColdFusion data source.
- Create banners that draw data (such as URLs or image file paths) out of a database.

The `CF.query` function can retrieve data from any supported ColdFusion data source (see "About ColdFusion data sources" on page 15).

About ColdFusion data sources

Your ColdFusion administrator can help you identify and configure data sources. To create ActionScript files that successfully perform queries on ColdFusion MX data sources, you msut know how the data source is identified by ColdFusion, as well as any other parameters that affect your ability to connect to that database, such as whether a username and password are required to connect. Your ColdFusion administrator can provide this information.

For ColdFusion developers, the term **data source** can refer to a number of different types of structured data accessible locally or across a network. You can query websites, LDAP servers, POP mail servers, and documents in a variety of formats.

However, for server-side ActionScript, a data source ordinarily means the entry point to a ColdFusion database.

For more detailed information about ColdFusion data sources, see *Administering ColdFusion Server* in the ColdFusion MX Server documentation.

Accessing ColdFusion data sources using ActionScript

The server-side ActionScript feature in ColdFusion MX provides the ability to return record set data to a Flash MX client from a ColdFusion data source. The ColdFusion data source name and the SQL statement you execute on the data source are both arguments you specify in the `CF.query` function in the server-side ActionScript.

Typically, your server-side ActionScript handles the interaction with the ColdFusion data source, and returns a record set to the Flash MX client through the Flash Remoting service.

For information about building queries using the `CF.query` function in server-side ActionScript, see "The CF.query ActionScript function" on page 16.

The CF.query ActionScript function

You use the `CF.query` in your server-side ActionScript to retrieve data from a ColdFusion data source. This function lets you perform queries against any ColdFusion data source.

Note: `CF.query` maps closely to the `cfquery` CFML tag, though it currently supports a subset of the `cfquery` attributes.

Use the `CF.query` function to do the following:

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

For reference information about the `CF.query` function, see Chapter 5, "ColdFusion ActionScript Functions" on page 29.

CF.query function syntax

You can write the `CF.query` ActionScript function using either named arguments or positional arguments. The named argument style is more readable than the positional argument style, even though it is more laborious to code `CF.query` declarations in this way. Although the positional argument style supports a subset of `CF.query` arguments, it allows a more compact coding style that is more appropriate for simple expressions of the `CF.query` function.

CF.query named argument syntax

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

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

Note: The named argument style requires curly braces {} to surround the function arguments.

CF.query positional argument syntax

The positional argument approach supports a subset of `CF.query` arguments, but it lets you code in a leaner more efficient style. The schema for the positional argument style is as follows:

```
// CF.query positional argument syntax
CF.query(datasource, sql);
CF.query(datasource, sql, maxrows);
CF.query(datasource, sql, username, password);
CF.query(datasource, sql, username, password, maxrows);
```

Note: When using positional arguments, do not use curly braces {}.

For more information about the `CF.query` function, see "CF.query," in Chapter 5.

The CF.query record set

The `CF.query` function returns a RecordSet object, which is an instance of the RecordSet class of objects. The RecordSet class provides a wide range of functions for handling record set data.

You use methods in the RecordSet ActionScript class in your client-side ActionScript to scrub, manipulate, mine, filter, sort, or otherwise change data returned in the `CF.query` record set.

Currently, the following methods are available in the RecordSet class:

Method	Description
addItem	Appends a record to the end of the specified RecordSet
addItemAt	Inserts a record at the specified index
addView	Requests notification of changes in a RecordSet object's state
filter	Creates a new RecordSet object that contains selected records from the original RecordSet object
getColumnNames	Returns the names of all the columns of the RecordSet
getItemAt	Retrieves a record from a RecordSet object
getItemID	Gets the unique ID corrresponding to a record
getLength	Returns the total number of records in a RecordSet object
getNumberAvailable	Returns the number of records that have been downloaded from the server
isFullyPopulated	Determines whether a RecordSet object can be edited or manipulated
isLocal	Determines whether a RecordSet object is local or server-associated
removeAll	Removes all records from the RecordSet object
removeItemAt	Removes a specified record
replaceItemAt	Replaces the entire contents of a record
setDeliveryMode	Changes the delivery mode of a server-associated record set
setField	Replaces one field of a record with a new value
sort	Sorts all records by a specified compare function
sortItemsBy	Sorts all the records by a selected field

These functions are available for every RecordSet object returned by the `CF.query` function to the Flash MX client. You invoke these functions as follows:

```
objectName.functionName();
```

For example, in the result function that you create to handle record set data returned by the `CF.query` function, you can reference the database column names returned in the record set using the `getColumnNames` RecordSet function:

```
function selectData_result ( result )
{
   //result holds the query data; employeesView is a Flash list box
   stringOutput.text = result.getColumnNames();
   _root.employeesView.setDataProvider(result);
}
```

Building a simple application

The following procedures describes how to build a simple server-side ActionScript application. The example application, a corporate personnel directory, uses the NetServices object to connect to the `personneldirectory` server-side ActionScript. The `personneldirectory` server-side ActionScript retrieves data from a ColdFusion data source and returns the results to the Flash movie as a RecordSet object.

Note: The server-side ActionScript that you create provides the backend services in an application.

This example requires the following:

- A server-side ActionScript file named personneldirectory.asr that includes functions that interact with a ColdFusion data source.
- A client-side Flash MX movie in which the NetServices object is created.

To create the application:

1. Write the server-side ActionScript that performs the database query and returns data back to the client through the Flash Remoting service.
2. Create the Flash movie interface.
3. Define a search function that sends user data to the Flash service.
4. Define a result function that captures the results returned from the Flash service.
5. Ensure that the Flash movie has established a connection to the Flash Remoting service.

For more information, see the following sections:

- "Creating the Flash movie interface" on page 21
- "Submitting user data to the Flash Remoting service" on page 21
- "Capturing Flash Remoting service results" on page 22
- "Checking for a Flash Remoting service connection" on page 22

Write the server-side ActionScript

In this example, you create a search function that performs a simple search operation against a ColdFusion data source. This function accepts two arguments, firstName and lastName, and returns any records found that match these parameters.

Create a server-side ActionScript file that contains the following code, and save the file as personneldirectory.asr:

```
//search takes firstName lastName arguments
function search(firstName, lastName)
{
   searchdata = CF.query({datasource: "bigDSN",
      sql:"SELECT * from personnel WHERE fname = firstName AND lname = lastName"{);

   if (searchdata)
      return searchdata;
   else
      return null;
}
```

Creating the Flash movie interface

The Flash movie interface consists of one frame with a variety of text boxes and a submit button.

To create the Flash movie interface:

1 In the Flash MX authoring environment, create a new Flash source file, and save it as pDirectory.fla.

2 Create two input text boxes. Name one text box variable **lastName** and the other **firstName**.

3 Create a dynamic text box, and name its variable **status**.

4 Insert a list box component, and name it **dataView**.

5 Insert a push button component.

6 Save your work.

The following figure shows what the pDirectory Flash movie looks like:

Submitting user data to the Flash Remoting service

To send data to a server-side ActionScript, you must create a function that passes the data from the Flash movie to the server-side ActionScript. The `search` function, applied at the frame level, collects the user-entered data from the firstName and lastName text boxes and passes the data as function arguments to the directoryService object, which is created when the Flash movie connects to the Flash Remoting service. For more information, see "Checking for a Flash Remoting service connection" on page 22.

The following is a Flash MX ActionScript example:

```
#include "NetServices.as"
function search()
{
   // The search() method is defined in the server-side AS file
   directoryService.search(firstName.text, lastName.text);
   dataView.setDataProvider(null);
   status.text = "waiting...";
}
```

In this example, the search function passes the contents of firstName and lastName text boxes to the server-side ActionScript.

The `dataView.setDataProvider(null)` function clears the `dataView` list box component. The `status.text` action displays a message in the status text box while the record set is being retrieved from the server-side ActionScript.

Capturing Flash Remoting service results

When you create a function that calls a server-side ActionScript function, you must also create a function to handle the data returned by the server-side ActionScript. You access data returned by a server-side ActionScript by defining a function with the same name as the function making the initial call, only you append `_Result` to the name.

For example, if you create a function called `basicQuery` to return query data, you also need to define a results function to handle returned data; the results function would be declared as `basicQuery_result`.

In the following example, the results function `search_Result` supplies the record set to the `dataView.setDataProvider` function:

```
function search_Result(resultset)
{
   dataView.setDataProvider(resultset);
   status.text = (0+resultset.getLength())+" names found.";
}
```

In this example, the `_Result` suffix tells the Flash Remoting service to return the results of the `search` function to this function. The `dataView.setDataProvider(resultset)` function assigns the results returned by the Flash Remoting service to the `dataView` list box. The `status.text` action displays the number of records returned by the Flash Remoting service.

Checking for a Flash Remoting service connection

To ensure that the Flash movie is connected to the Flash Remoting service, you use an `if` statement; for example:

```
if (inited == null)
{
   inited = true;
   NetServices.setDefaultGatewayUrl("http://localhost:8100/flashservices/
      gateway");
   gateway_conn = NetServices.createGatewayConnection();
   directoryService = gateway_conn.getService(personneldirectory, this);
   status.text = "Type into the text boxes, then click 'Search'";
}
```

In this example, `inited` is evaluated for a value. If it is `null` (not connected), the movie connects to the Flash Remoting service using the NetServices object. For more information about connecting to the Flash Remoting service, see "Connecting to the Flash Remoting service" on page 8.

CHAPTER 4
Using the CF.http Function

This chapter describes how to use the CF.http ActionScript tag to conduct HTTP (HyperText Transfer Protocol) operations.

Contents

Overview

The `CF.http` ActionScript function lets you to 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 `CF.http` 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.

Using CF.http to retrieve HTML content

The most basic way to use the `CF.http` function is to use it with the `Get` method argument to retrieve a page from a specified URL. For the `CF.http` function, the `Get` method is the default.

For example, the following server-side code retrieves file content from the specified URL:

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

On the client side, your code could look like the following:

```
#include "NetServices.as"
NetServices.setDefaultGatewayUrl("http://mycfserver:8100");
gatewayConnnection = NetServices.createGatewayConnection();
myHttp = gatewayConnnection.getService("httpFuncs", this);

// This is the server-side function invocation
url = "http://anyserver.com";
myHttp.basicGet(url);

// Create the results function
function basicGet_result()
{
   url = "http://anyserver.com
   ssasFile.basicGet(url)
}
```

Data returned by the CF.http function

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 available properties:

Property	Description
Text	A Boolean value indicating whether the specfied URL location contains text data.
Charset	The charset used by the document specified in the URL. HTTP servers normally provide this information, or the charset is specified in the charset parameter of the Content-Type 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, macromedia.com 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 content types include "text/html", "image/png", "image/gif", "video/mpeg", "text/css", and "audio/basic".
Responseheader	Response header. If there is one instance of a header key, this value can be accessed as a simple type. If there is more than one instance, values are put in an array in the responseHeader structure.

Statuscode	HTTP error code and associated error string. Common HTTP status codes returned in the response header include the following:
	`400: Bad Request`
	`401: Unauthorized`
	`403: Forbidden`
	`404: Not Found`
	`405: Method Not Allowed`

Referencing HTTP Post parameters CF.http

In order to pass HTTP `Post` parameters in the `CF.http` function, you must construct an array of objects and assign this array to a variable named `params`. The following arguments can only be passed as an array of objects in the `params` argument of the `CF.http` function.

Parameter	Description
name	The variable name for data that is passed
type	Transaction type: • URL • FormField • Cookie • CGI • File
value	Value of URL, FormField, Cookie, File, or CGI variables that are passed

In the following example, the `CF.http` function passes HTTP `Post` parameters in an array of objects.

```
function postWithParamsAndUser()
{
   // Setup the array of post parameters. These are just like cfhttpparam tags.
   params = new Array();
   params[1] = {name:"arg2", type:"URL", value:"value2"};

   url = "http://localhost:8100/";

   // Invoke with the method, url, params, username, and password
   result = CF.http("post", url, params, "karl", "salsa");
   return result.get("Filecontent");
}
```

Using the CF.http Post method

Use the `Post` method to send cookie, form field, CGI, URL, and file variables to a specified ColdFusion page or CGI program for processing. For POST operations, you must use the `params` argument for each variable that you post. The `Post` method passes data to a specified ColdFusion page or an executable that interprets the variables being sent and returns data.

For example, when you build an HTML form using the `Post` method, you specify the name of the page to which form data is passed. You use the `Post` method in `CF.http` in a similar way. However, with `CF.http`, the page that receives the Post does not display anything.

```
function postWithParams()
{
   // Setup the array of post parameters. These are just like cfhttpparam tags.
   // This example passes formfield data to a specified URL.
   params = new Array();
   params[1] = {name:"Formfield1", type:"FormField", value:"George"};
   params[2] = [name:"Formfield2", type:"FormField", value:"Brown"};

   url = "http://localhost:8100/";

   // Invoke CF.http with the method, url, and params
   result = CF.http("post", url, params);
   return result.get("Filecontent");
}
```

Using the CF.http Get method

You use the `Get` method to retrieve files, including text and binary files, from a specified server. You reference properties of the object returned by the `CF.http` function to access things like file content, header information, MIME type, and so on.

For more information about `CF.http` function properties, see Chapter 5, "ColdFusion ActionScript Functions" on page 29.

Example using the Get method

The following example of using the `CF.http` function shows a common approach to retrieving data from the web.

```
// Returns content of URL defined in url variable
// This example uses positional argument style
function get()
{
   url - "http://www.macromedia.com/software/coldfusion/";

   //Invoke with just the url argument. Get is the default.
   result = CF.http(url);
   return result.get("Filecontent");
}
```

CHAPTER 5
ColdFusion ActionScript Functions

This chapter explains the syntax and usage of the two currently supported server-side ActionScript funtions, `CF.query` and `CF.http`.

Contents

CF.query

Description — Performs queries against ColdFusion data sources.

Return value — Returns a RecordSet object. For information about methods that you can use to manipulate the record set returned by the `CF.query` function, see "Methods available in the RecordSet ActionScript class" on page 31.

Syntax

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

Arguments

Arguments	Req/Opt	Description
datasource	Required	Name of the data source from which query retrieves data.
sql	Required	SQL statement.
username	Optional	Overrides the username specified in the data source setup.
password	Optional	Overrides the password specified in the data source setup.
maxrows	Optional	Maximum number of rows to return in the record set.
timeout	Optional	Maximum number of seconds for the query to execute before returning an error indicating that query has timed out. Can only be used in named arguments.

Usage — You can code the CF.query function using named arguments or using positional arguments. You can invoke all supported arguments using the named argument style, as follows:

```
CF.query({datasource:"datasource", sql:"sql stmt",
  username:"username", password:"password", maxrows:"maxrows",
  timeout:"timeout"});
```

Note: The named argument style uses curly braces {} to surround the function arguments.

Positional arguments let you use a shorthand coding style. However, not all arguments are supported for the positional argument style. Use the following schemas to code the `CF.query` function using positional arguments:

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

Note: When using positional arguments, do not use curly braces {}.

Example

```
// Define a function to do a basic query
// Note use of positional arguments
function basicQuery()
{
   result = CF.query("myquery", "cust_data", "SELECT * from tblParks");
   return result;
}

// Example function declaration using named arguments
function basicQuery()
{
   result = CF.query({datasource:"cust_data", sql:"SELECT * from tblParks"});
   return result;
}

// Example of query function using maxrows argument
function basicQueryWithMaxRows()
{
   result = CF.query("cust_data", "SELECT * from tblParks", 25);
   return result;
}

// Example of CF.query with username and password
function basicQueryWithUser()
{
   result = CF.query("cust_data", "SELECT * from tblParks",
      "wsburroughs", "migraine1");
   return result;
}
```

Methods available in the RecordSet ActionScript class

The record set returned by the `CF.query` function can be manipulated using methods in the RecordSet ActionScript class. The following are some of the methods available in the RecordSet class:

- `RecordSet.getColumnnames`
- `RecordSet.getLength`
- `RecordSet.getItemAt`
- `RecordSet.getItemID`
- `RecordSet.sortItemsBy`
- `RecordSet.getNumberAvailable`
- `RecordSet.filter`
- `RecordSet.sort`

For more detailed information about the RecordSet ActionScript class, see *Using Flash Remoting*.

CF.htttp

Description

Executes HTTP POST and GET operations on files. POST operations upload MIME file types to a server, or post cookie, formfield, URL, file, or CGI variables directly to a server.

Return value

Returns an object containing properties that you reference to access data. For available properties returned by the `CF.http` function, see "Properties available with the CF.http object" on page 34.

Syntax

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

Arguments

Arguments	Req/Opt	Description
method	Required	Two arguments are supported: • get: downloads a text or binary file or creates a query from the contents of a text file. • post: sends information to the server page or CGI program for processing. Requires the `params` argument.
url	Required	The absolute URL of the host name or IP address of server on which the file resides. The URL must include the protocol (http or https) and host name.
username	Optional	When required by a server, a username.
password	Optional	When required by a server, a password.

Arguments	Req/Opt	Description
resolveurl	Optional	For `Get` and `Post` methods. • Yes or No. Default is No. For GET and POST operations, if Yes, page reference that is returned into the Filecontent property has its internal URLs fully resolved, including port number, so that links remain intact. The following HTML tags, which can contain links, are resolved: - img src - a href - form action - applet code - script src - embed src - embed pluginspace - body background - frame src - bgsound src - object data - object classid - object codebase - object usemap
params	Optional	HTTP parameters passed as an array of objects. Supports the following parameter types: • name • type • value `CF.http` params are passed as an array of objects. The `params` argument is required for POST operations. For detailed information about each params argument, see "CF.http parameters" on page 34 .
path	Optional	The path to the directory in which to store files. When using the path argument, the `file` argument is required.
file	Optional	Name of the file that is accessed. For GET operations, defaults to name specified in the `url` argument. Enter path information in the `path` attribute.

Usage

You can code the `CF.http` function using named arguments or positional arguments. You can invoke all supported arguments using the named argument style, as follows:

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

Note: The named argument style uses curly braces {} to surround the function arguments.

Positional arguments let you use a shorthand coding style. However, not all arguments are supported for the positional argument style. Use the following schemas to code the `CF.http` function using positional arguments:

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

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

Note: When using positional arguments, do not use curly braces {}.

CF.http parameters

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

Parameter	Description
name	The variable name for data that is passed
type	The transaction type: • URL • FormField • Cookie • CGI • File
value	Value of URL, FormField, Cookie, File, or CGI variables that are passed

Properties available with the CF.http object

The `CF.http` function returns data as a set of object properties:

Property	Description
Text	A Boolean value that indicates whether the specfied URL location contains text data.
Charset	The charset used by the document specified in the URL. HTTP servers normally provide this information, or the charset is specified in the charset parameter of the Content-Type 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, macromedia.com 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 content types include text/html, image/png, image/gif,"video/mpeg, text/css, and audio/basic.

Property	Description
Responseheader	Response header. If there is one instance of a header key, value can be accessed as simple type. If there is more than one instance, values are put in an array in responseHeader structure.
Statuscode	HTTP error code and associated error string. Common HTTP status codes returned in the response header include: 400: Bad Request 401: Unauthorized 403: Forbidden 404: Not Found 405: Method Not Allowed

You access these attributes using the `get` function:

```
function basicGet()
{
    url = "http://localhost:8100/";

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

Example The following examples show a number of the ways to use the `CF.http` function:

```
function postWithNamedArgs()
{
   // Set up the array of post parameters.
   params = new Array();
   params[1] = {name:"arg1", type:"FormField", value:"value1"};
   params[2] = {name:"arg2", type:"URL", value:"value2"};
   params[3] = {name:"arg3", type:"CGI", value:"value3"};

   url = "http://localhost:8100/";

   path = application.getContext("/").getRealPath("/");
   file = "foo.txt";

   result = CF.http({method:"post", url:url, username:"karl", password:"salsa",
          resolveurl:true, params:params, path:path, file:file});

   if (result)
     return result.get("Statuscode");
   return null;
}

// Example of a basic HTTP get operation
// Shows that HTTP Get is the default
function basicGet()
{
   url = "http://localhost:8100/";

   // Invoke with just the URL. This is an HTTP Get.
   result = CF.http(url);
   return result.get("Filecontent");
```

```
}

// Example showing simple array created to pass params arguments
function postWithParams()
{
   // Set up the array of post parameters. These are just like cfhttpparam tags.
   params = new Array();
   params[1] = {name:"arg2", type:"URL", value:"value2"};

   url = "http://localhost:8100/";

   // Invoke with the method, url, and params
   result = CF.http("post", url, params);
   return result.get("Filecontent");
}

// Example with username and params arguments
function postWithParamsAndUser()
{
   // Set up the array of post parameters. These are just like cfhttpparam tags.
   params = new Array();
   params[1] = {name:"arg2", type:"URL", value:"value2"};

   url = "http://localhost:8100/";

   // Invoke with the method, url, params, username, and password
   result = CF.http("post", url, params, "karl", "salsa");
   return result.get("Filecontent");
}
```