

# ADOBE® FLEX® Gumbo Preview Release Tutorials

© 2009 Adobe Systems Incorporated. All rights reserved.

Adobe® Flex® 4 Features and Migration Guide. Prerelease version.

This prerelease version of the Software may not contain trademark and copyright notices that will appear in the commercially available version of the Software.

Adobe, the Adobe logo, Flash, Flex, Flex Builder and LiveCycle are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. ActiveX and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Apple and Macintosh are trademarks of Apple Inc., registered in the United States and other countries. Linux is a registered trademark of Linus Torvalds. Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Solaris is a registered trademark or trademark of Sun Microsystems, Inc. in the United States and other countries. All other trademarks are the property of their respective owners.

This Work is licensed under the Creative Commons Attribution Non-Commercial 3.0 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/3.0/>

Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704, USA.

# Contents

**Chapter 1: Building a basic application**

**Chapter 2: Creating ColdFusion services for client applications**

**Chapter 3: Querying a database**

**Chapter 4: Flash Builder Code Generation for Accessing Services**

**Chapter 5: Manage the access of data through paging**

Paging (ColdFusion) ..... 14

Paging tutorial (PHP) ..... 15

**Chapter 6: Using data management to synchronize server updates**

Data Management (ColdFusion) ..... 18

# Chapter 1: Building a basic application

This tutorial illustrates the basic concepts for creating, building, and running applications using Flash Builder and the Flex framework.

## Create and run an application

Create an application that contains a text item.

- 1 In Flash Builder, select File > New > Flex Project.
- 2 Specify Hello for the project name.
- 3 Make sure that the following default settings are correctly specified:
  - Application type: Web
  - Flex SDK Version: Use default SDK
  - Application Server Type: None/Other
- 4 Click Finish.

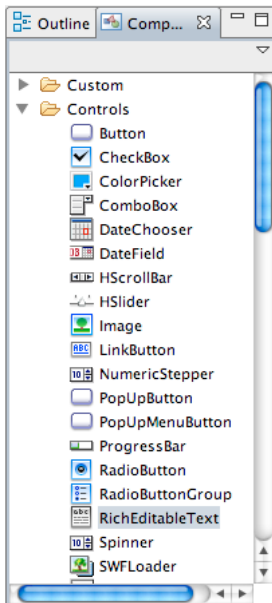
Flash Builder creates the project and opens an empty MXML application source file in Source View.

Flex projects always contain application source files. An MXML file is a source file providing an interface to an application. The Flex project also contains compiled versions of the application, libraries, and other assets used by the application.

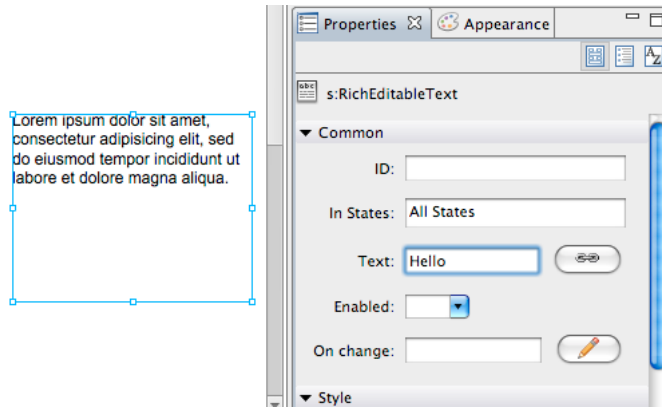
- 5 Select Design.

Flash Builder switches to Design View.

- 6 In the Components View, make sure that the Controls folder is open. Scroll the controls to find the RichEditableText control.

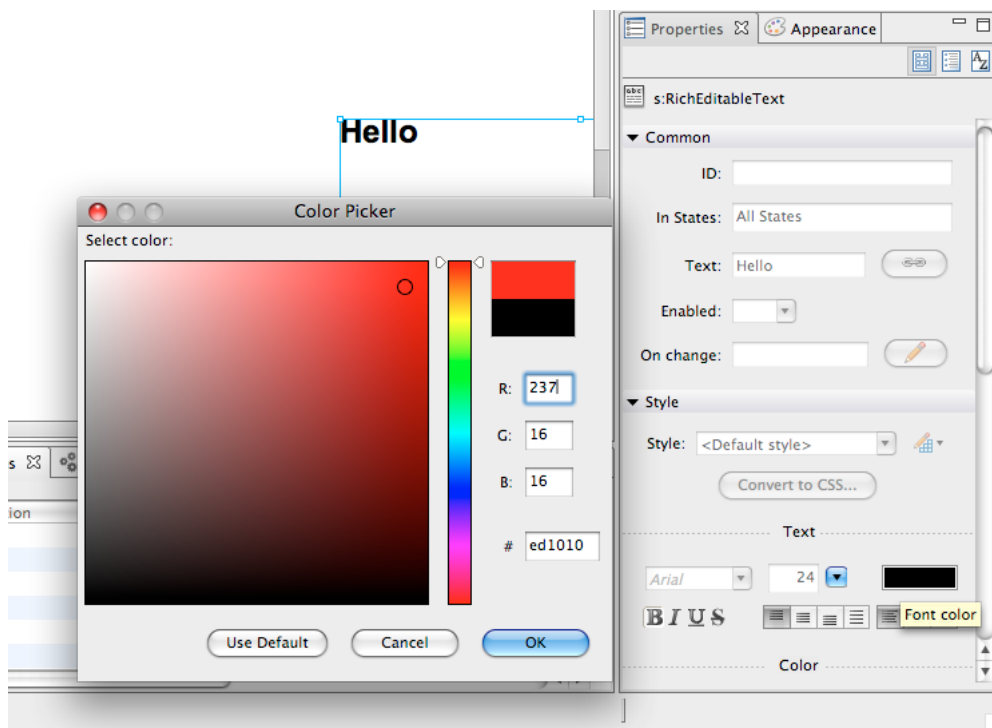


- 7 Drag a RichEditableText component from the Components View to the editing area of Design View.
- 8 Make sure the RichEditableText component is still selected. In the Property Inspector, select the Text field and replace the default text with **Hello**.



- 9 In the Property Inspector, modify the appearance of the text by specifying the following:

Font Size	24
Font Attribute	Bold
Font Color	Select a shade of red from the Color Picker



Specifying attributes for a Text control (Mac OS)

- 10 Select Source to switch to Source View.

A RichEditableText tag has been inserted into the MXML file. The RichEditableText tag contains attributes for the values you specified in Design View.

- Place the cursor after the color attribute in the RichEditableText tag. Type a space. Then begin typing **sel** to add the selectionColor attribute.

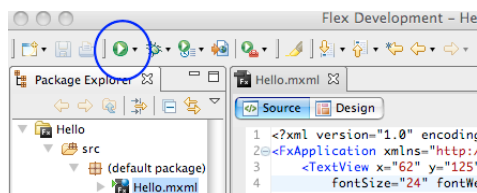
A list of available attributes updates as you type. If you click an attribute in the list, documentation for the attribute appears.

Select selectionColor and specify: #005555



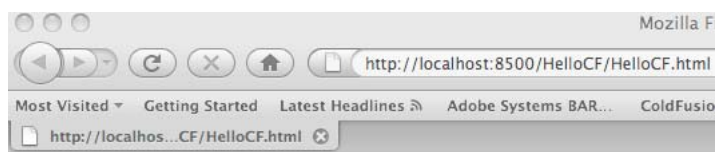
Code hinting in Flash Builder

- Save the file, and then click the Run button to run the application.



Run button

The application runs in a web browser.



Select the text in the web browser to view the color of selected text. Then close the browser.

**Locate problems in source code and create a release version of the application**

Introduce an error in an application. Then find and fix the error. Finally, export a release version of the application.

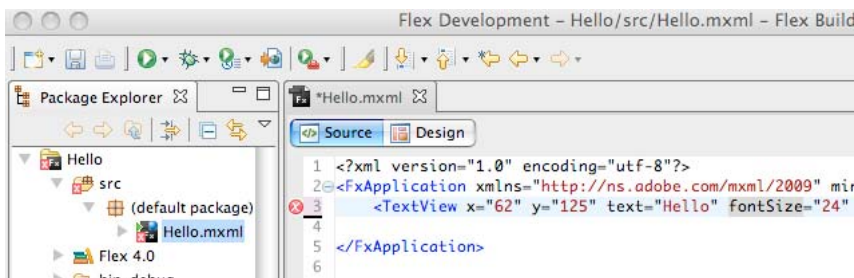
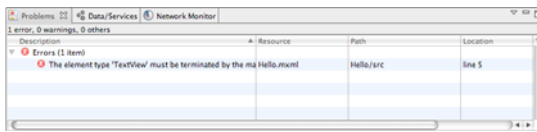
- 1 In the Hello.mxml file created in the previous procedure, delete the closing angle bracket in the RichEditableText tag.

Change /> to /.

- 2 Save the file.

When you save a file, Flash Builder automatically compiles the application. Flash Builder flags errors in the Package Explorer and source code. Flash Builder lists errors in the Problems View.

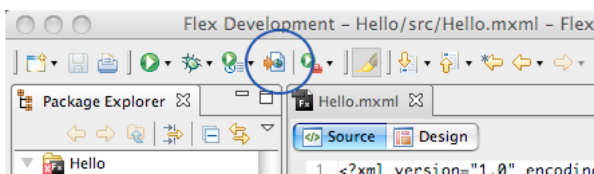
The Problems View is now visible. Expand the Errors list to view the error.



- 3 Double-click the error to go directly to the source code. Fix the error and click Save.

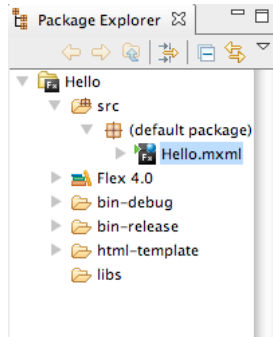
There are no errors listed in Problems View.

- 4 Select the Export Release Build button to create a release version of the application.



- 5 In the Export Release Build dialog box, select the Enable View Source check box. Accept the default settings and click Finish.

Flash Builder creates a bin-release folder and places the application and its assets in this folder. The application is ready to be deployed.



During application development, Flash Builder compiles the application into a SWF file that contains debug information. The default location for the SWF file is the bin-debug folder in your Flex project. You can export a release version of the application, which does not contain the debug information. The default location for a release version is the bin-release folder in your Flex project.

### **Summary**

This tutorial shows the basics of creating a Flex project and building an application using both Source View and Design View of the source editor. It shows how to locate errors in source code and the difference between a debug and release version of an application.

# Chapter 2: Creating ColdFusion services for client applications

You can create applications with Flex that access services from a wide variety of technologies. These technologies include:

- ColdFusion
- PHP
- ASP.NET
- Ruby on Rails

This tutorial shows how to implement and test a basic ColdFusion service. A subsequent tutorial shows how to access the remote service from Flex. Another tutorial shows how to bind data returned by the service to Flex components in the client application.

The principles of creating and testing ColdFusion services can be applied to other service technologies.

## Create a ColdFusion component

Create a ColdFusion component (CFC) that can be accessed as a service by Flex over the Internet.

### 1 Code the following CFC:

```
<cfcomponent>
    <cffunction name="getMessage" access="remote" returntype="string">
        <cfset message = "Hello from CF!">
        <cfreturn message>
    </cffunction>
</cfcomponent>
```

The function `getMessage` is a service operation that client applications access.

Make sure to specify `remote` as the access type for the function.

### 2 Save the CFC as `HelloService.cfc` in your web root. Place it in a folder named `HelloCF`.

```
<CF Web Root>/HelloCF/HelloService.cfc
```

### 3 In `HelloService.cfc`, add operations that return localized messages and locales.

```
<cffunction name="getLocalizedMessage" returnType="string" access="remote">
  <cfargument name="locale" required="yes"/>

  <cfswitch expression="#locale#">
    <cfcase value="en">
      <cfset message = "Hello from CF!" />
    </cfcase>
    <cfcase value="es">
      <cfset message = "Hola de CF!" />
    </cfcase>
    <cfcase value="ne">
      <cfset message = "Namaskar CF bata!" />
    </cfcase>
    <cfdefaultcase>
      <cfthrow message="Unknown locale code" />
    </cfdefaultcase>
  </cfswitch>
  <cfreturn message>
</cffunction>

<cffunction name="getLocales" returnType="array" access="remote">
  <cfset codes = listToArray("en,es,ne") />
  <cfreturn codes>
</cffunction>
```

The `getLocalizedMessage` function specifies an argument of type string. The `getLocales` function returns an array of strings.

### Test the Hello ColdFusion service

There are various ways to test the service before you access it from an application.

- 1 Create a ColdFusion script file, `helloservicetest.cfm`, that calls the service and displays the returned data in HTML.

Use the `cfinvoke` tag to call the service.

Use the `cfdump` tag to view the results.

Testing `getMessage()`...

```
<cfinvoke component="HelloService" method="getMessage" returnvariable="message"/>
<p> Result: <cfdump var="#message#" /> </p>
```

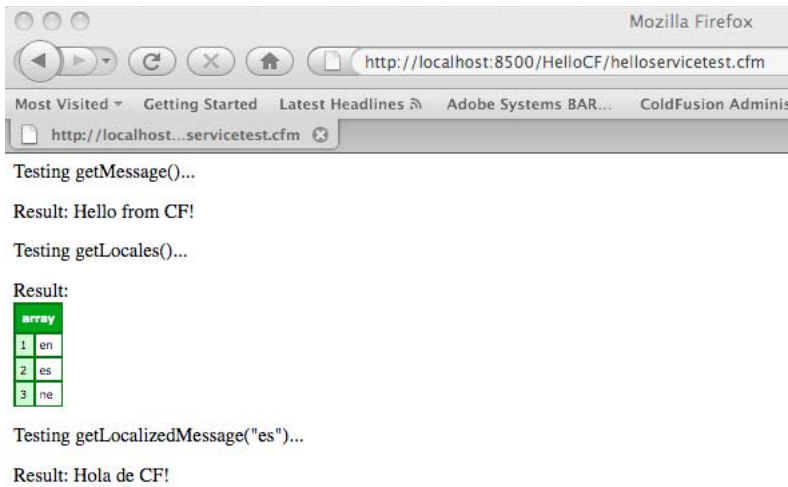
Testing `getLocales()`...

```
<cfinvoke component="HelloService" method="getLocales" returnvariable="locales"/>
<p> Result: <cfdump var="#locales#" /> </p>
```

Testing `getLocalizedMessage("es")`...

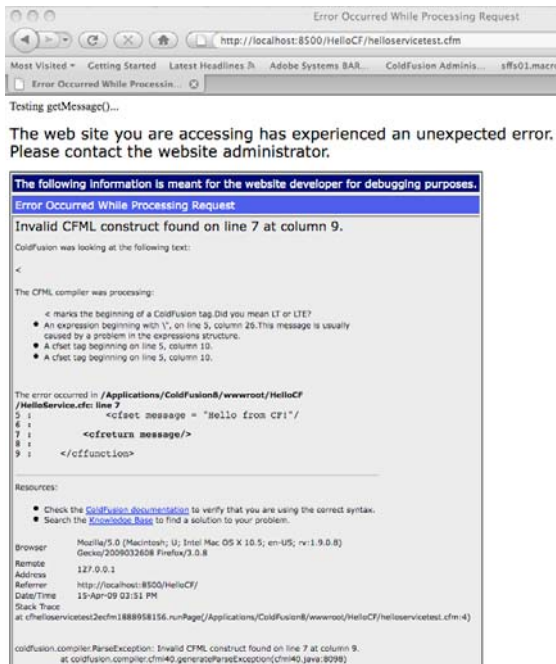
```
<cfinvoke component="HelloService" method="getLocalizedMessage" locale="es"
returnvariable="message"/>
<p> Result: <cfdump var="#message#" /> </p>
```

- 2 Save `helloservicetest.cfm` in the same folder as `HelloService.cfc`, and call the service from a web browser.



*Using cfinvoke and cfdump*

If there is an error in the service, ColdFusion displays detailed information about the error.



You can also enable Robust Exception Information for the ColdFusion server to get detailed debugging output.

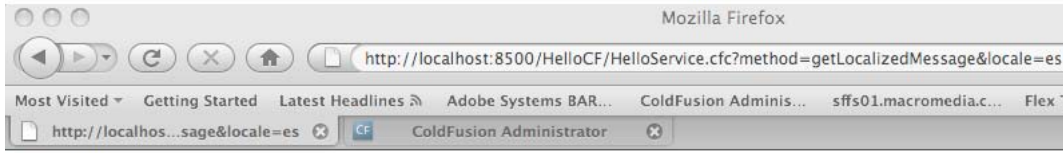
**3** Add trace statements.

Use ColdFusion trace statements as necessary to debug your service.

**4** Call a function in the service directly from a URL.

For example, call the getLocalizedMessage function with the parameter es:

```
http://localhost:8500/HelloCF/HelloService.cfm?method=getLocalizedMessage&locale=es
```



Hola de CF!

*Testing a service directly from a URL*

## Chapter 3: Querying a database

This tutorial shows how to access a remote database and display records retrieved from the database in a DataGrid.

The tutorial illustrates the following concepts:

- Using a ColdFusion component (CFC) to implement service operations that query a database.
- Binding a CFC service operation to a Flex data control, such as a DataGrid.
- Sampling the data returned from a service operation to configure a data type for the returned data.
- Running the application and viewing the returned data in the DataGrid.

### Querying a database with Flash Builder

This tutorial accesses the cfbookclub database that is provided with a ColdFusion installation. It lists the contents of the Books table in the database. However, you can modify this tutorial to select records from any database you might have access to.

- 1 In your favorite ColdFusion editor, create the following CFC.

This CFC selects all the records from the Books table in the cfbookclub database.

```
<cfcomponent>
  <cffunction name="getBooks" access="remote" returntype="query">
    <cfquery name="qBooks" datasource="cfbookclub">
      SELECT * FROM Books
    </cfquery>
    <cfreturn qBooks>
  </cffunction>
</cfcomponent>
```

- 2 Save the CFC as BookService.cfc in your web root. Place it in a folder named books.

```
<CF Web Root>/books/BookService.cfc
```

- 3 Test the CFC before implementing it in an application.

It is always a good idea to test a service before importing it into a Flex project.

- a Create the following ColdFusion script to test the CFC. This script uses cfdump to view the results of the service operation.

```
Testing getBooks()...

<cfinvoke component="BookService" method="getBooks" returnvariable="books"/>


<p> <cfdump var="#books#"/>
```

- b Save the script as bookservicetest.cfm in the books folder next to BookService.cfc.
  - c From a web browser, run the script to test the service.
- 4 In Flash Builder, create a ColdFusion server project and name it Books.
 

Be sure to set the Application Server Type to ColdFusion. Enable Use Remote Object Access Service and select ColdFusion remoting.

Specify the folder containing the service as the output folder for the project.
  - 5 In Design View, add a DataGrid control to the application.

- 6 In the Property Inspector, click the Data Provider button.
- 7 In the No Services Defined dialog, click Yes to connect to a service.
- 8 In the New Flex Service wizard, select ColdFusion. Click Next.

 *Flash Builder provides multiple ways to connect to a data service. In this scenario, you first create the user interface. From a user interface component, you can connect to a service and specify the remote operation.*

- 9 Specify Import CFC and navigate to the BookService.cfc you created previously. Click Finish.  
Provide authorization credentials as needed for your system.

The Data Services View now displays the BookService.

- 10 With the DataGrid selected, again click the Data Provider button in the Property Inspector.

The Bind to Data dialog opens with New Service Call selected.

BookService is the only service available in the Flex project.

getBooks() is the only operation available in the service.

- 11 In the Bind to Data dialog, select Configure Return Type to define the data type for returned data.


Flex needs to know the return data type to access service operations. The BookService service does not specify the data type for returned data. Flash Builder uses client-side typing to define a custom data type for the returned data.

- 12 Select Create a New Custom Data type and specify Book for the name of the type. Click Next.

The BookService returns a complex data type representing a database record for a book. The custom type Book provides access to each field of the record.

- 13 The getBooks operation does not require any arguments. Click Next.

- 14 View the properties of the Book data type returned by the service. Click Finish.

 *When Flash Builder configures a return type it accesses the database to create a value object. The properties of the custom data type are derived from the value object. You can view and modify the properties of the data type before proceeding.*

- 15 Click OK.

Flash Builder binds the data returned from the service call to the DataGrid component. It modifies the columns of the DataGrid, binding the value returned for each Book property to a column in the DataGrid.

- 16 Make sure the DataGrid is still selected. In the Property Inspector, click Configure Columns and then do the following steps:

- a Select the ISSPOTLIGHT column. Click Delete to delete the column.
- b Similarly, delete all columns except TITLE and GENRE.
- c Select the TITLE column. Scroll to the headerText property and rename the column Title.
- d Similarly, rename the GENRE column to Genre.

- 17 In Design View, resize the DataGrid to a more normal shape, then save and run the application.

# Chapter 4: Flash Builder Code Generation for Accessing Services

In the previous tutorial, Flex generated client code that accessed a ColdFusion service.

There are three parts to the generated code:

- Event handler
- Service call and responder
- Data binding

This tutorial walks you through the Flex code generated in the previous tutorial. There are no “steps” to this tutorial. Just a description of the code generation.

## Event handlers

Flex uses event handlers to trigger a service call.

Flex code is not processed “top to bottom,” as in server templates. Instead, applications built with Flex listen for events to trigger responses.

By default, Flex uses the `creationComplete` event on a component to trigger a service call.

Examine the `TextView` component in the `HelloCF.mxml` source code:

```
<TextView x="70" y="60"
    fontSize="28" fontWeight="bold" color="#B10C0C"
    text="{getMessageResult.lastResult}"
    creationComplete="getMessageResult.token = helloService.getMessage()" />
```

The `creationComplete` event fires after the application creates the `TextView` component. For this `TextView` component, `creationComplete` calls the `getMessage` operation from the service.

Other events, such as a `Click` event on a button or a `selectionChange` event of a list, can be used to trigger service calls.

## Service calls and CallResponders

When you add a service, Flex creates a component containing methods that call the service operations. When you invoke one of the methods, the component calls the corresponding service operation over the network using the appropriate protocol.

Service calls are associated with `CallResponders`. Data returned from a service call is placed in the `lastResult` attribute of the `CallResponder`. `CallResponders` allow service calls to operate asynchronously. The client application can continue to be responsive while the service call is being processed.

Examine the generated service call and responder:

```
<Declaration>
  <CallResponder id="getMessageResult"/>
  <helloservice:HelloService id="helloService"
    fault="Alert.show(event.fault.faultString) "
    showBusyCursor="true"
    destination="coldFusion"
    endpoint="http://localhost:8500/flex2gateway/"
    source="HelloCF.HelloService"/>
</Declaration>
```

When you imported the HelloService service, Flex created the HelloService class. This class contains methods to invoke the getMessage(), getLocales(), and getLocalizedMessage operations of the service.

The CallResponder, identified by the getMessageResult id, provides access to the returned data.

### Data-binding code

Flex data binding allows you to assign the value of the lastResult attribute of a CallResponder to a UI component. When the last result updates, Flex automatically updates the component.

Flex uses curly braces { } to bind data to an attribute of a component. In this example, Flex binds the lastResult value to the text attribute of the TextView component.

Examine the data binding for the TextView component:

```
<TextView x="70" y="60"
  fontSize="28" fontWeight="bold" color="#B10C0C"
  text="{getMessageResult.lastResult}"
  creationComplete="getMessageResult.token = helloservice.getMessage() "/>
```

The data returned to the CallResponder, getMessageResult.lastResult, is automatically assigned as the text attribute for the TextView component.

### Summary

In summary, Flash Builder generates code to access the imported service in the client application:

- After Flex creates the TextView component in the running application, the creationComplete event fires, calling a service operation.
- The data returned from the service call is placed in the lastResult attribute of the associated CallResponder.
- The text attribute of the TextView, which is bound to the lastResult attribute, updates with the returned data.

For more information, refer to:

- BULLET\_ITEM
- BULLET\_ITEM
- BULLET\_ITEM

# Chapter 5: Manage the access of data through paging

For various performance and network bottleneck issues, you typically do not retrieve all the records in a database with a single call. Paging allows you to retrieve the records incrementally in sets, and only on an as needed basis.

To use paging to retrieve records, your server code must implement two functions with the following signatures:

- `getItems_paged($startIndex, $numItems)`
- `count()`

## Paging (ColdFusion)

The application you create in this tutorial accesses the employee database that you previously downloaded and installed.

This tutorial creates an application that uses paging to populate a DataGrid.

### Create the remote service and import it into a Flex project

- 1 In your favorite ColdFusion editor, create a ColdFusion service that implements the required paging functions.

```
<cfcomponent output="false">
    <cffunction name="getItems_paged" output="false" access="remote" returntype="any" >
        <cfargument name="startIndex" type="numeric" required="true" />
        <cfargument name="numItems" type="numeric" required="true" />

        <cfset var qRead="">
        <cfquery name="qRead" datasource="employees">
            SELECT * FROM employees LIMIT #startIndex#, #numItems#
        </cfquery>

        <cfreturn qRead>

    </cffunction>

    <cffunction name="count" output="false" access="remote" returntype="numeric" >

        <cfquery name="qread" datasource="employees">
            SELECT COUNT(emp_no) AS empCount FROM employees
        </cfquery>

        <cfreturn qread.bookCount>

    </cffunction>
</cfcomponent>
```

- 2 Save the service in the web root of your ColdFusion installation.

Name the file `PagingService.cfc` and place it in a directory named `PagingCF`.

- 3 In Flash Builder, create a Flex project. Name the project PagingCF and specify ColdFusion for the server technology as listed below. Click Next.
  - Application Server Type: ColdFusion
  - Enable Use Remote Object Access Service
  - Select ColdFusion Flash Remoting
- 4 Validate your ColdFusion settings and specify the PagingCF directory for the Output Folder. Click Finish.
- 5 From the Flash Builder Data menu, select Connect to Data Service. Select ColdFusion and click Next.
- 6 Verify that Import CFC is selected. Browse to the PagingService.cfc file you created in step 1. Click Finish.
- 7 In the Flash Builder Data/Services view, from the context menu for the getItem\_paged() operation, select Configure Return Type.
- 8 Create a custom data type named Employee. Click Next.
 

Flex uses custom data types to access and update complex data returned from a server.
- 9 Specify values and types for the parameters to the getItem\_paged() operation as described below. Click Next:

Argument	Argument Type	Value
startIndex	Number	0
numItems	Number	10

- 10 View the properties of data returned from the service. Click Finish.
- 11 From the context menu for the getItem\_paged operation, select Enable Paging.
- 12 In the Select Unique Identifier dialog, select emp\_no and click Next.
- 13 In the Confirm Paging dialog, specify the count() operation from the drop-down list and click Finish

## Create an application and bind getItem\_paged() to a DataGrid

- 1 Select the Design tab to open Design View for the MXML Editor.
- 2 From the Components view, drag a DataGrid component onto the Design Area and place it near the top.
 

The DataGrid component is available under Data Controls.
- 3 Make sure the DataGrid component is selected, then in the Flex Properties view change the Editable attribute to False.
- 4 With the DataGrid component selected, from the Data menu select Bind to Data.
- 5 In the Bind to Data operation, select New Service Call, then select the getItem\_paged() operation from the drop-down list, and click OK.
- 6 Save and run the file.
 

In the application, click in the DataGrid scroll bar or move the thumb of the scroll bar to observe the paging of data.

## Paging tutorial (PHP)

The application you create in this tutorial accesses the employee database that you previously downloaded and installed.

This tutorial creates an application that uses paging to populate a DataGrid.

## Create the remote service and import it into a Flex project

- 1 In your favorite PHP editor, create a PHP service class that implements the required paging functions.

```
<?php
class PagingService {

private function connect() {
    $connection = mysql_connect("YOUR MYSQL SERVER", "USERNAME", "PASSWORD") or
die(mysql_error());
    mysql_select_db("employees", $connection) or die(mysql_error());
}

public function getItems_paged($startIndex, $numItems) {
    $this->connect();
    $startIndex = mysql_real_escape_string($startIndex);
    $numItems = mysql_real_escape_string($numItems);
    $sql = "SELECT * FROM employees LIMIT $startIndex, $numItems";
    $result = mysql_query($sql) or die('Query failed: ' . mysql_error());
    return $result;
}

public function count() {
    $this->connect();
    $sql = "SELECT * FROM employees";
    $result = mysql_query($sql) or die('Query failed: ' . mysql_error());
    $rec_count = mysql_num_rows($result);
    mysql_free_result($result);
    return $rec_count;
}
}
?>
```

- 2 Modify the connect function to provide your server, username, and password for access to your database.
- 3 Save the service in the web root of your PHP installation, as described below.
  - In your web root directory, create a directory named PagingPHP.
  - In the PagingPHP directory, create a directory named services.
  - Name the file PagingService.php and save it int services directory.
- 4 In Flash Builder, create a Flex project. Name the project PagingPHP and specify PHP for the server technology. Click Next.
- 5 Click Next. Specify the Web Root and Root URL for your system. Validate your server settings and specify the PagingPHP directory for the Output Folder. Click Finish.
- 6 From the Flash Builder Data menu, select Connect to Data Service. Select PHP and click Next.
- 7 Verify that Import PHP Class is selected. Browse to the PagingService.php file you created in step 1. Click Finish.  
If the Zend Framework, which includes Zend AMF, is not installed on your system, click OK to install the Zend Framework.
- 8 In the Flash Builder Data/Services view, from the context menu for the getItems\_paged() operation, select Configure Return Type.
- 9 Create a custom data type named Employee. Click Next.

Flex uses custom data types to access and update complex data returned from a server.

10 Specify values and types for the parameters to the `getItems_paged()` operation as described below. Click Next:

Argument	Argument Type	Value
startIndex	Number	0
numItems	Number	10

11 View the properties of data returned from the service. Click Finish.

12 From the context menu for the `getItems_paged` operation, select Enable Paging.

13 In the Select Unique Identifier dialog, select `emp_no` and click Next.

14 In the Confirm Paging dialog, specify the `count()` operation from the drop-down list and click Finish

## Create an application and bind `getItems_paged()` to a DataGrid

1 Select the Design tab to open Design View for the MXML Editor.

2 From the Components view, drag a DataGrid component onto the Design Area and place it near the top.

The DataGrid component is available under Data Controls.

3 Make sure the DataGrid component is selected, then in the Flex Properties view change the Editable attribute to False.

4 With the DataGrid component selected, from the Data menu select Bind to Data.

5 In the Bind to Data operation, select New Service Call, then select the `get Items Paged()` operation from the drop-down list, and click OK.

6 Save and run the file.

In the application, click in the DataGrid scroll bar or move the thumb of the scroll bar to observe the paging of data.

## Chapter 6: Using data management to synchronize server updates

Data management features allow you to synchronize adding, updating, and deleting of records in a database. Changes you make in the client application are not written to the server until a commit method is called. You can call a revert method to roll back changes made in the client application.

To implement data management, your server code must implement a combination of functions with the following signatures:

- `createItem(item:CustomDatatype):int`
- `deleteItem(itemID:Number):void`
- `updateItem((item: CustomDatatype):void`
- `getItem(itemID:Number): CustomDatatype`

*CustomDatatype* is a data type representing complex data returned from the server. In server-side typing, the service defines the custom data type. In client-side typing, you introspect the service to configure the custom data type.

### Data Management (ColdFusion)

The application you create in this tutorial accesses the employee database that you previously downloaded and installed.

In this tutorial, you create an application that contains an editable DataGrid displaying employee records. You can modify one or more records in place in the DataGrid, and also add or delete selected records from the DataGrid. All the changes in the DataGrid are local until you select a Save All Changes button, which updates the database.

#### Create the remote service and import it into a Flex project

- 1 In your favorite ColdFusion editor, create a ColdFusion service that implements the required data management functions.

```

<cfcomponent output="false">
    <cffunction name="getAllItems" output="false" access="remote" returntype="any" >
        <cfset var qAllItems="">
        <cfquery name="qAllItems" datasource="fb_tutorial_db">
            SELECT * FROM employees
        </cfquery>
        <cfreturn qAllItems>
    </cffunction>

    <cffunction name="getItem" output="false" access="remote" returntype="any" >
        <cfargument name="itemID" type="numeric" required="true" />

        <cfset var qItem="">
        <cfquery name="qItem" datasource="fb_tutorial_db">
            SELECT * FROM employees
                WHERE emp_no = <CFQUERYPARAM CFSQLTYPE="CF_SQL_INTEGER"
VALUE="#ARGUMENTS.itemID#">
        </cfquery>

        <cfreturn qItem>
    </cffunction>

    <cffunction name="createItem" output="false" access="remote" returntype="any" >
        <cfargument name="item" required="true" />

        <cfquery name="createItem" datasource="fb_tutorial_db" result="result">
            INSERT INTO employees (first_name, last_name, gender, birth_date,
hire_date)
                VALUES (<CFQUERYPARAM cfsqltype="CF_SQL_VARCHAR"
VALUE="#item.first_name#">,
                <CFQUERYPARAM cfsqltype="CF_SQL_VARCHAR"
VALUE="#item.last_name#">,
                <CFQUERYPARAM cfsqltype="CF_SQL_VARCHAR" VALUE="#item.gender#">,
                <CFQUERYPARAM cfsqltype="CF_SQL_DATE" VALUE="#item.birth_date#">,
                <CFQUERYPARAM cfsqltype="CF_SQL_DATE" VALUE="#item.hire_date#">)
        </cfquery>

        <cfreturn result.GENERATED_KEY/>
    </cffunction>

    <cffunction name="updateItem" output="false" access="remote" returntype="void" >
        <cfargument name="item" required="true" />

        <cfquery name="updateItem" datasource="fb_tutorial_db">
            UPDATE employees SET birth_date = <CFQUERYPARAM cfsqltype="CF_SQL_DATE"
VALUE="#item.birth_date#">,
    
```

```

        hire_date = <CFQUERYPARAM cfsqltype="CF_SQL_DATE" VALUE="#item.hire_date#">,
        gender = <CFQUERYPARAM cfsqltype="CF_SQL_VARCHAR" VALUE="#item.gender#">,
        first_name = <CFQUERYPARAM cfsqltype="CF_SQL_VARCHAR"
VALUE="#item.first_name#">,
        last_name = <CFQUERYPARAM cfsqltype="CF_SQL_VARCHAR" VALUE="#item.last_name#">
        WHERE emp_no = <CFQUERYPARAM CFSQLTYPE="CF_SQL_INTEGER" VALUE="#item.emp_no#">
    </cfquery>
</cffunction>

<cffunction name="deleteItem" output="false" access="remote" returntype="void" >
    <cfargument name="itemID" type="numeric" required="true" />

    <cfquery name="delete" datasource="fb_tutorial_db">
        DELETE FROM employees
        WHERE emp_no = <CFQUERYPARAM CFSQLTYPE="CF_SQL_INTEGER"
VALUE="#ARGUMENTS.itemID#">
    </cfquery>
</cffunction>

</cfcomponent>

```

- 2 Save the service in the web root of your ColdFusion installation.

Name the file EmployeeService.cfc and place it in a directory named DataMgtCF.

- 3 In Flash Builder, create a Flex project. Name the project DataMgtCF and specify ColdFusion for the server technology as listed below. Click Next.
  - Application Server Type: ColdFusion
  - Enable Use Remote Object Access Service
  - Select ColdFusion Flash Remoting
- 4 Validate your ColdFusion settings and specify the DataMgtCF directory for the Output Folder. Click Finish.
- 5 From the Flash Builder Data menu, select Connect to Data Service. Select ColdFusion and click Next.
- 6 Verify that Import CFC is selected. Browse to the EmployeeService.cfc file you created in step 1. Click Finish.
- 7 In the Flash Builder Data/Services view, from the context menu for the getItem(), select Configure Return Type.
- 8 Create a custom data type named Employee. Click Next.
 

Flex uses custom data types to access and update complex data returned from a server.
- 9 Specify a value and type for the parameters to the getItem() operation as described below. Click Next:

Argument	Argument Type	Value
itemID	Number	100000

- 10 View the properties of data returned from the service. Click Finish.
- 11 In the Flash Builder Data/Services view, from the context menu for the getAllItems() operation, select Configure Return Type.
- 12 Use an existing data type. Select Employee[] from the drop-down list. Click Finish.

## Enable Data Management Features

- 1 In the Data/Services view, expand the Data Types node for EmployeeService and select the Employee data type.

- 2 From the context menu for the Employee data type, select Enable Data Management.
- 3 In the Select Unique Identifier dialog, select emp\_no and click Next.
- 4 In the Map Database Operations dialog, specify the following operations and click Finish.
  - **Add (Create) Operation:** createItem( item: Employee )
  - **Get All Properties Operation:** getItem( itemID: Number )
  - **Update Operation:** updateItem( item: Employee )
  - **Delete Operation:** deleteItem ( itemID: Number )

Data management is now enabled for this operation. Flash Builder generates client code that can update data using a combination of the mapped operations.

## Create the application and add a DataGrid and Buttons

- 1 Select the Design tab to open Design View for the MXML Editor.
- 2 From the Components view, drag a DataGrid component onto the Design Area and place it near the top.  
 The DataGrid component is available under Data Controls.
- 3 In the Flex Properties view, with the DataGrid selected, type “dg” for the ID property.
- 4 Drag four Buttons to the Design Area, lining them up beneath the DataGrid.
- 5 Double-click each button to edit their labels. Provide the following Labels:

Label
Add
Delete
Revert
Save All Changes

- 6 In the Data/Services view, select the getAllItems() operation and drop it onto the DataGrid.
- 7 (Optional) Click Configure Columns to rename and rearrange the columns.

## Generate and code event handlers for the Buttons

Each Button needs an event handler to specify the action to take when the Button is clicked. Flash Builder generates stubs for event handlers, which you can then code to specify the service actions to take.

- 1 Select the Add button. Then, for the On Click field of the Add button, click the Pencil icon and then select Generate Event Handler.  
 The MXML editor changes to Source View, placing the cursor in the generated event handler.
- 2 In the Script block, add the following import statement:
 

```
import services.employeeservice.Employee;
```
- 3 In the event handler body, type the following:

```

var e:Employee = new Employee();
var birthDate:Date = new Date(2000, 01, 01);
var hireDate:Date = new Date(2000, 01, 01);

e.first_name = "New";
e.last_name = "New";
e.birth_date = birthDate;
e.hire_date = hireDate;
e.gender = "M";
dg.dataProvider.addItem(e);
dg.verticalScrollPosition = dg.dataProvider.length -1;

```

As you type, Flash Builder content assist helps you view the available methods and values.

- 4 Modify the DataGrid columns for birth\_date and hire\_date to add a date chooser:

```

<DataGridColumn headerText="birth_date" editorDataField="selectedDate"
    dataField="birth_date" itemEditor="mx.controls.DateField"/>

```

```

<DataGridColumn headerText="hire_date" editorDataField="selectedDate"
    dataField="hire_date" itemEditor="mx.controls.DateField"/>

```

- 5 In Design View, add an On Click event handler for the Delete button and specify the following code:

```
employeeService.deleteItem(dg.selectedItem.emp_no);
```

- 6 Similarly, add an On Click event handler for the Revert button with the following code:

```
employeeService.getDataManager
    (employeeService.DATA_MANAGER_EMPLOYEE).revertChanges();
```

- 7 Add an On click event handler for the Save All Changes button with the following code:

```
employeeService.commit();
```

- 8 Save the application and select Run > Run DataMgtCF.

You can update and delete selected employees in the DataGrid. You can also add new employees.

Click the Revert button to undo any changes you made.

Click the Save All Changes button to write all changes to the database.

