Flex in a Week, Flex 4.5

Video2.07_remote: Retrieving and handling data with RemoteObject

You now know how to retrieve data from a remote server using the Flex framework's HTTPService and WebService components.

In this video, I will first introduce you to the Remoting Service and then show you how to use the Flex framework's RemoteObject component to request data using the service.

You will learn how to access remote methods to request data and then use the result and fault events to handle the data or errors.

The remoting service allows a client application to access methods of server-side code.

Communication between the client and the server is formatted using the Action Message Format, or AMF.

AMF is a binary format for data transfer that is significantly smaller and faster than equivalent XML-based message formats.

The Flex framework is compatible with servers that have version 3 of the AMF protocol.

You can use Adobe LiveCycle Data Services, or the open-source BlazeDS product for Java implementations.

You can also use ColdFusion MX 7.0.2 or later to communicate with CFCs.

PHP developers also have several options for working with AMF.

The remoting service architecture exchanges data between the Flex client and an AMF-compatible server.

The server-side remote object service, that sends and receives AMF messages, is known as Flash Remoting.

The client-side component is an ActionScript class in the Flex framework named RemoteObject.

Remember that this training series is software-agnostic, since we focus on the Flex code, which is the same regardless of the backend technology.

If you want to explore more server-specific information, remember that you can do so by accessing the Adobe Developer Connection and then selecting one of the technologies for Flex integration.

In an earlier video, we compared the file size for the same data returned using the HTTPService, WebService and RemoteObject components and we found that the binary data in AMF was half the size of HTTPService data and a third of the size of the WebService data.

This smaller footprint makes RemoteObject smaller and faster than the other two RPC components.

Using the RemoteObject component, you can also access server methods directly from your client-side code.

You will learn about value objects later in this day of training, but if you are creating a strongly typed object on the server, you can create an equivalent ActionScript object in the Flex application or you can have Flash Builder generate it for you.

The implementation also translates the server language data types into ActionScript data types and vice versa.

For example, here are the links to the serialization tables for ActionScript to Java, and ActionScript to web services, in the LiveCycle Data Services documentation.

When I click on one of the links, you can see the data conversion details.

If your Flex application is on the same server as the remoting service, then you must define a service destination, which is essentially an alias mapping to a single service, in a file named remoting-config.xml on the server.

You will define the following.

An id attribute, which is an arbitrary name for referencing the service from within your Flex client application.

The source property describes the fully qualified package or path to the service being called.

This code shows an example of a service destination for a Java class.

The id property is just an alias you can use to reference this java class from the RemoteObject component in the Flex application.

The source property describes the fully qualified package and name of the class to be called.

This code also includes a scope property indicating the scope of the service.

It could be set to application, session or request.

In this training, I assume that you are working with Flash Builder on your local environment, but are accessing the adobetes.com server remotely.

Therefore, you cannot configure the server's remoting-config.xml file.

Instead, you will set the service destination at compile time on the RemoteObject MXML tag.

The destination value will be specific to the backend server's implementation.

For the previous example, the destination would be the javaClassAlias defined in the remoting-config.xml file on the server.

Since this training relies on a ColdFusion server, you will set the destination to ColdFusion.

This value is set in the remoting-config.xml file on that server.

The service that you will be accessing is a ColdFusion Component (CFC) named employeeData in the specified source property path.

The endpoint property points to the address for the remoting gateway on the server.

To use the RemoteObject MXML tag, you declare it in a Declarations tag block.

Remember that the Declarations block is where you instantiate non-visual MXML components in your MXML application code.

You define an instance id property for the object and then register the result and fault event handlers.

For a RemoteObject instance, you must invoke the remote service method in a separate statement.

You call the operation as a method of the RemoteObject instance using dot notation.

You reference the service object's id property, the dot, and then the method name.

Obviously, you will need to know the server-side method names.

You can invoke these operations on Flex framework system and user events.

This example invokes the operation on the creationComplete event of the Application container.

This example demonstrates the service method being called on the click event of a Button instance.

Here is the Employee Portal: Vehicle Request Form that you created in Day 1 and modified in earlier exercises on this Day of training.

Note that the DropDownList control does not contain any data because I have removed the HTTPService object from the Declarations block.

I will replace it with a RemoteObject instance in a moment, but let's first review the rest of the code.

You can see the UI components to create the form at the bottom of the file in the UI components section.

You can also see the functions that add the DateChooser event listeners and handle the validation on those controls.

At the top of the Script block, you can see the employees class variable that is an instance of the ArrayCollection class and that is bound to the DropDownList control as its dataProvider.

Remember that it is this employees property that I need to populate with the return data from the remote service method call.

Between the Declarations tags, I am adding a RemoteObject instance with an id property set to employeeService.

I am adding a destination property and setting the value to ColdFusion.

The source property's value is f45iaw100 – to reference the name of this series – .remoteData.employeeData.

The endpoint property value is this URL (http://adobetes.com/flex2gateway/).

I want to request the data when the application loads, so I am locating the opening Application container tag.

You can see that the creationComplete event of the instance calls the initApp() method, which adds the event listeners for the DateChooser component instances.

Below the event listeners, I am typing employeeService.getEmployees() to invoke the remote service operation.

I am saving the file and then selecting the Network Monitor view and enabling the tool.

I am running the application and then switching back to Flash Builder to look at the network traffic.

The RemoteService call is for the getEmployees() method.

I am selecting it and then clicking on the Response tab to the right.

When I expand the Response body, you can see that the type is AMF and that all the data is indexed.

Note that the property names are all in uppercase letters.

Like the HTTPService object, if the data is returned as an array of objects, then Flex will convert it to an instance of the ArrayCollection class.

Also like the HTTPService object, you can directly access the data through the lastResult property.

The syntax for remote services uses dot syntax starting with the service object's id property.

Then you reference the remote method name and then the lastResult property itself.

If the service call returned a simple string and the service operation is named employeeService.getEmployeeRecord(), then the string is accessed by typing employeeService dot getEmployeeRecord – without the parentheses to denote the method – dot lastResult.

As with HTTPService, you can exert more control over the returned data if you handle it in a result event rather than simply binding to the lastResult property.

You handle the result event like you would any other event on a component in the Flex framework.

Here the result event is placed on the component with a defined event handler, which passes the event object as the one argument.

The event object is typed to the ResultEvent class, which you must import for use.

The ResultEvent class data types its result property as an object.

If you try to assign the result data to an ArrayCollection variable, you will get an implicit coercion error.

Therefore, you must use the as operator to cast the result property as an instance of the ArrayCollection class.

In my code, I am adding a result event on the RemoteObject instance and then generating the result handler using code assist.

I am Control + clicking on the handler name to locate it in the Script block.

Here is the event object data typed to the ResultEvent class and here is the imported class, added by Flash Builder.

Remember that I want to place all the returned server data into this employees property, which is an ArrayCollection instance.

Inside the result event handler, I am typing employees and then setting the property equal to event.result.

When I save the file, you can see the implicit coercion error.

I am using the as operator to cast the event.result property as an ArrayCollection instance.

When I save the file, the compiler error goes away.

Now, I’m scrolling down to find the DropDownList control.

Remember that the ColdFusion server sent back all the properties in uppercase letters.

ColdFusion is not case-sensitive and does this conversion automatically.

To compensate, I am changing the labelField’s LASTNAME property to all uppercase letters and doing the same for the PHONE property.

When I save the file and run the application, you’ll see that the data is displayed as expected.

Handling faults for the RemoteObject component is very similar to handling results.

A fault event will dispatch when there are problems retrieving data from a service.

A fault event will also dispatch when the requestTimeout property is exceeded.

The fault event also dispatches an event object, so you will have access to properties like target, type, fault and others.

As when handling a result event, you will define an event listener function and pass the event object to it.

The FaultEvent class contains four String properties.

The faultDetail property contains extra details about the fault.

The faultCode property is a simple code for describing the fault.

The faultString property is a text description of the fault.

The message property is a concatenation of the other three properties.

I am modifying the source property of the RemoteObject instance to reference a path that doesn’t exist.

When I save the file and run the application, you can see that the application displays a runtime error.

Back in the main application code, I am adding a fault event to the RemoteObject instance and then generating the event handler.

When I save the file and run the application again, you can see that simply having a fault handler, even if it’s empty, will prevent the fault from being displayed.

In the fault handler, I am creating an alert dialog by typing Alert.show() and then passing the event.fault.faultString property for the first argument.

This will display the faultString in the body of the message.

I’m setting the second parameter to “Fault Information” for display in the dialog title bar.

Note that Flash Builder did automatically import the Alert class for me.

When I save the file and run the application the Alert dialog box will appear with the fault message.

When you need to define different event handlers for multiple methods of a single remote object service, use the method MXML tag nested inside of the RemoteObject tags.

The name property consists of the server-side method name.

The result and fault events define the respective event handlers.

If you do not define result or fault handlers in the method MXML tag, then the application will default to the ones defined in the RemoteObject instance.

For your next step, work through the RemoteObject exercise titled “Populating an application with data and handling faults”.