Flex in a Week, Flex 4.5

Video 2.07_web: Retrieving and handling data with WebService

So far in this training, you have used the HTTPService component to communicate with a server to retrieve both static and dynamic XML data.

In this video, you will learn how to use the Flex framework's WebService component to invoke web service methods.

You will also learn that handling result and fault events for WebService objects is similar to handling them for HTTPService objects.

Lastly, you will learn how to call multiple methods from the same WebService object.

Flex applications can interact with SOAP-based web services.

SOAP, or the Simple Object Access Protocol, is an industry-standard specification for web services.

The SOAP interface is defined in an XML-based Web Service Description Language (WSDL) document.

The WSDL must be accessible at runtime using a URL accessible over HTTP.

In Flex, you use the WebService component to access this SOAP-based service.

The component does runtime introspection of a WSDL to determine how to make and receive the messages from the server.

To connect to a web service, you first declare the WebService MXML component in a Declarations tag block.

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

You should define an id property for the WebService object to uniquely identify the component instance and to make it accessible to your ActionScript code.

Next you must define the wsdl property, which is a relative or absolute address to the web service.

The address can refer to static or dynamically generated WSDL documents.

In either case the document will be loaded automatically when the WebService object is created.

In the code I will create for this video, you will see that I access a WSDL document that is dynamically generated by a remote ColdFusion server.

Remember that this training series uses ColdFusion to generate all the example data, but you can use PHP, Java, .NET or any server technology that

can generate web services.

Note that, unlike the HTTPService object definition which directly accessed the data, this WebService instance only accesses the WSDL document, not the methods on the server that directly provide the data.

For a WebService object, you must invoke the web service method – also called an operation – in a separate statement.

You call the operation as a method of the WebService object 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, which you can find in the WSDL.

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

For example, this code calls the getEmployeeRecord() operation of the employeeService object on the WebService component's load event.

This event is triggered when the WSDL document has loaded successfully.

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

Note that if the WebService object has not loaded the WSDL document by the time the creationComplete event is dispatched, then the call will be queued and executed when the WSDL has loaded.

This last 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 WebService object 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 dataProvider of the DropDownList control.

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

This is the WSDL document that I will load into the application.

I am scrolling down the XML file to find the getEmployees() operation.

This is the method I will invoke to grab all of the employee data from the server.

Between the Declarations tags, I am pressing CTRL+Space twice, so that the content assist tool shows the available code templates, and typing We and selecting the WebService code template.

Now I am removing the closing WebService tag and modifying the opening tag so that it uses single tag syntax instead of tag block syntax.

I am changing the code template so that the WebService object's id property is set to employeeService and I am pasting the URL for the WSDL document into the wsdl property.

We will use the result and fault events later in this video, but for now remove all of the properties in the WebService object that we have not modified.

Remember that this code simply loads the WSDL document, it does not retrieve any data.

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 web 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.

Note that there are two requests that have been logged.

The first one is an HTTPService call to load the WSDL document.

The second one is the web service request for the getEmployees() method.

I’m clicking on the Response tab to the right.

When I expand the Response body tree, you can see all the employee data in the XML format.

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 web services, however, uses dot syntax starting with the service object's id property.

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

Here is an example of this code, if the service call returned a simple string.

If the service operation is named employeeService.getEmployeeRecord(), then the string is accessed by typing employeeService dot getEmployeeRecord – without the parentheses – 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 mx.rpc.events.ResultEvent class, which you must import.

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 WebService object and then using the Flash Builder code assist tool to generate the result handler.

I am Control + clicking on the handler name to locate it in the Script block to see that the event object is data typed to the ResultEvent class.

If I scroll up a bit, you can see that Flash Builder automatically imported the class for me.

Remember that the employees property is the ArrayCollection object that I want to place all of the server data into.

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 that the compiler throws a coercion error stating that the result property of the event object is an Object and cannot be associated with the employees property, which is an ArrayCollection instance.

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

When I save the file, the compiler is now happy.

Scrolling down to the UI components section of my code, I can see that the DropDownList control is bound to the employees property and the lastName field is registered as the data field to display in the control.

When I run the application, unfortunately, the data shows Object Object.

This is happening because the DropDownList control cannot locate the lastName property of the returned data.

To debug this problem, I am placing a breakpoint on the closing curly brace of the result handler and then Debugging the application.

When prompted, I am switching to the Debug perspective and then maximizing the Variables tab to drill down into the event object.

You can see that property names are all spelled in uppercase.

This is because ColdFusion is not case sensitive and so converts all variables to uppercase letters.

ActionScript, however, is case sensitive, so back in my code, after I stop the debug session and return to the Flash perspective, I am updating the labelField to use the LASTNAME property, spelled in all uppercase.

I am doing the same to the PHONE property in the TextInput control for the Office Phone field.

When I save the file and run the application, you can see the data now populates the DropDownList control and the phone number also reflects the correct number.

Handling faults for the WebService 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 wsdl property of the WebService object to reference a URL that doesn't exist.

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

I am returning to the main application code to register a fault event on the WebService object.

I am using Flash Builder to generate the event handler code, which you can see is generated in the Script block.

When I save the file and run the application again, you can see that the application doesn't display the data, but it doesn't display the runtime error either.

You don't actually need to have anything in the fault handler.

Simply having one will prevent errors from being displayed.

I am adding a breakpoint to the closing brace of the fault handler and then debugging the application.

In the Variables view, you can see that the event object has a fault object with all of the fault properties, including this faultString.

I am terminating the debugging session and then returning to the Flash perspective.

In the fault handler, I am calling the Alert.show() method and passing the event.fault.faultString property for the value that will be displayed in the Alert dialog body text.

I am also passing the words "Fault Information" for the dialog titlebar.

I also need to make sure that Flash Builder did import the Alert class for me, which it did.

When I save the file and run the application, you can see that the Alert dialog box appears.

You call multiple methods from the same WebService object by invoking them on the same or different events

As I discussed earlier, when you use the returned data, you must reference the service operation name in the reference using dot notation.

The problem of invoking multiple methods on the same service arises when you want to specify different result and/or fault handlers for each method.

When the result and fault events are associated with the WebService tag, rather than individual operations, all operations use the same event handlers.

You use the operation tag to define multiple operations for your web service.

The operation tag is a compiler tag, which does not directly correspond to ActionScript objects or properties.

Other examples of compiler tags include Style and Declarations.

The operation tag must be nested inside of a WebService tag block.

It requires a unique name property, which corresponds to the method being invoked on the server.

You can define different result and fault handlers on each operation tag, but usually the result handlers will be different and the fault handler will use the parent definition.

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