Flex in a Week, Flex 4.5
Video 2.07_http: Retrieving and handling data with HTTPService

You have already used the HTTPService component in this training to retrieve static XML data from the server.

We did this on the first Day, to give you a sense of how to work with data in Flex.

In this video, I will start by reviewing some of the important points for working with the HTTPService component.

Then, we will replace the static XML file with a request for dynamically generated XML.

Lastly, you will learn how to handle faults, when the data is not successfully retrieved.

This is the Employee Portal: Vehicle Request Form that you created, laid out, styled and populated with data in Day 1.

Earlier in this second Day of training, you used event handling on the DateChooser controls to implement the validation logic to ensure that the return date is after the pickup date.

In the last exercise, you created this employees ArrayCollection instance and, again, used an event, but this time on the HTTPService object, to listen for the returned data from the server.

In the result event handler, you assigned the data to the employees class property, which is bound to the DropDownList control for display.

In this next section, I will formally review the details of this implementation.

Using the HTTPService component in the Flex framework, you can retrieve and send data using GET or POST requests to a URL over HTTP or HTTPS.

In Flex 4, you create the HTTPService object in a Declarations tag set since the object is not a visual UI component.

Remember that creating the object does not request the data.

You use the HTTPService object's send() method to retrieve the data, usually based on a user or system event.

After you create the HTTPService object in the Declarations block, you should assign it an instance name using the id property so that you can uniquely reference the object.

The second property you must set is url, which references the relative or absolute address for the remote data file.

The URL can reference a simple XML file stored relatively on the same file system, or one stored remotely on another server.

It can also access a static XML file or a dynamically built one via an application server like JSP, ColdFusion, .NET or PHP.

Back in the HTTPService implementation that you have already created, you can verify that the HTTPService object is inside of a Declarations block and has an id property set to employeeService.

The employeeService.send() method, which will actually request the data, is called from the initApp() function, which, is the event handler for the Application container's creationComplete event.

This means that as soon as the Flex application is initialized and created, it will request this data from the server.

Note that the HTTPService url property references this static XML file.

Here is the XML data in browser.

Instead of using a static XML file, I want to use a data-driven, dynamically generated XML file.

For that, I will access this URL.

As I mentioned in the last video, this training series will use a ColdFusion server for the server-side processing, but you could easily use PHP, Java, .NET or other servers to do the same thing.

The important point is that the server can generate the XML data on the fly and provide it to you via a URL.

This page looks like a long string of text, but if I view the source in the browser, you can see that it's all XML content that has been dynamically generated.

Back in the main application file, I am removing the static XML file and replacing it with this dynamic XML file.

The data that is returned in the HTTPService request is placed into the object's lastResult property.

You can request that data by referencing the HTTPService object's instance name plus the lastResult property.

In this case, that would be employeeService.lastResult.

In Day 1 of this training, you populated a DropDownList control with the employeeService.lastResult property.

Specifically, you used the repeating node, employees.employee, to reference all of the employees.

You also defined the labelField property to reference the lastName field of the XML data as the property to display in the control.

XML data retrieved through the HTTPService object is, by default, converted into a tree of ActionScript objects.

This is how we have, and will continue to, use it in this series.

However, you can use the resultFormat property to change the way that the data is handled in Flex.

You can review these formats by accessing the Flex Language Reference in Flash Builder Help

Based on your previous experience, you know that the returned data is represented as an ArrayCollection instance of generic objects.

You also learned that it is often more useful to handle the data with a result event rather than directly binding to the lastResult property.

Doing so allows you to manipulate the data before you use it or assign it to a class property, where it can be used more flexibly in the application.

Remember that the event object is of type ResultEvent.

This class must be imported before you use it.

Lastly, don’t confuse the lastResult property with the result property of the event object.

You use the lastResult property in data bindings directly to the returned results, and you use the result property in an event handler when you pass the event object to the event handler function.

Handling faults with HTTPService 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 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 url property of the HTTPService 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 HTTPService object.

When I use Flash Builder to generate the event handler code, you can see that new function 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 have to do anything in a 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 and the fault object properties are listed here.

Note the faultString property.

I am terminating the debugging session and 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 title.

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

When I save the file and run the application, you can see that the Alert dialog appears with the fault string and the Fault Information header.

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