Flex in a Week, Flex 4.5

Video 1.11: Requesting and retrieving XML data from the server

In Days 2 and 3 of this training series, you will learn, in depth, how to communicate with application servers to retrieve dynamic data.

In this video, I will show you a very simple way to retrieving XML data from a remote server.

You will also learn how to work with the Network Monitor to view your data request and the returned data.

Lastly, you will bind the return data to UI components in your display.

This is the Employee Portal: Vehicle Request Form that I have laid out and skinned earlier in this Day of training.

I have removed the skin, video and other visual styles to simplify the code for this video and related exercise.

Now you will learn how to retrieve data from an XML file to populate the Employee DropDownList control and then fill the Office Phone field with the employee's number.

In the videos and exercises that you have seen so far, most of the MXML components that you have encountered have been visual objects.

However, not all MXML components are visual.

For instance, the MXML components to retrieve data are not visual.

All non-visual objects in the 2009 specification for MXML must be created in a Declarations block.

Notice that the Declarations block is part of the fx namespace.

The Flex framework uses three primary approaches to send and receive data through remote connections.

The HTTPService component can retrieve and send data using the HTTP protocol Get and Post operations.

The WebService component works with WSDL documents via SOAP-based web services.

The RemoteObject component sends and retrieves data using the binary Action Message Format.

In this video, I will show you how to use HTTPService to retrieve XML data from a URL.

You will learn more about all three of these approaches in Days 2 and 3 of this training.

Remember that I am following a coding practice in these videos and exercises that places the code in specific locations.

This is the main starter application file for this example.

I am locating the Declarations tag block and adding an HTTPService component inside it.

I am adding an id property with a value of employeeService to the component in order to uniquely identify it.

I will discuss object naming more in the object oriented programming video, but for now note that I am naming this variable with an initial lowercase letter and then using camel case for the rest of the name.

This convention is common in many programming languages for naming objects, rather than classes.

Now I am adding the url property to point to the remote XML file (http://adobetes.com/f45iaw100/remoteData/employees.xml).

I am closing out the MXML tag with a forward slash and then an angle bracket.

.

Let's take a moment to look at this XML file.

You can see that this XML file contains all of the employees in this fictional company, with detailed information about each employee in the individual employee node.

If I was to run the application now, the HTTPService object that I defined here, would be instantiated, but no data will be loaded.

To load the data you will first need to know a little about events.

An event is a signal from the application to the Flash Player to perform some action.

There are two types of events.

A system event is dispatched by the Flex framework based on some logic that you define in your code or on the built-in functionality of a component.

A user event is dispatched when a user interacts with the application.

For instance, a button click or a mouse over action is a user-triggered event.

You will learn more about user and system events at the beginning of Days 2 and 3.

For this particular example, I will use a system event to retrieve the XML data.

I have located the Application tag and am adding a creationComplete event.

All UI components have a creationComplete event that is automatically triggered by the Flex framework when the component has been created in the UI and is ready for use.

In this case, when the Flex application has been created, I want it to access the employeeService instance of the HTTPService class that I created earlier.

Note that this value matches the id property of the HTTPService instance.

Specifically, I will run the send() method for that instance.

The send() method is a method of the HTTPService class that will initiate the retrieval of the XML data when the application is created.

This code is actually called the event handler for the creationComplete event.

Placing code inline is a common practice when you only have one line of ActionScript code in your event handler.

However, if you will need to provide more than one command, you can create an event handler function in ActionScript and reference it from the event.

Again, you will learn about that in Day 2.

Let’s use the Network Monitor to confirm that the data request was sent and retrieved by the application.

First, I have to double-click on the Editor tab to minimize the view.

I am selecting the Network Monitor view and then clicking the Enable Monitor button.

When I click the Run button, the application will display in the browser as usual.

I am returning to Flash Builder and double-clicking the Network Monitor view's tab to maximize the view.

Next I am selecting the recording of the HTTPService call and then clicking on the Request tab to see details about the service request.

Now I am clicking on the Response tab, and again drilling down, to reveal the employee data that is being returned.

The data structure displayed here matches the employees.xml file you viewed in the first section.

I am double-clicking the Network Monitor to minimize the view.

You learned about data binding in the last video when you bound the text property of the firstName and lastName TextInput controls to some static text to generate an email address.

In this step, you will use a data binding to attach the employee data from the HTTPService call to the DropDownList control.

Here is the DropDownList control instance.

I am adding the dataProvider property to the control and then adding the curly braces for the data binding.

Next I am referencing the HTTPService object, which is named employeeService.

The Flex framework actually provides all the data in a property of the service object named lastResult, which I am typing next.

Then you access the data by the XML nodes.

Back in the Network Monitor, you can see that each employee is accessed by first opening the employees node and then each individual employee node.

So, here in the binding, you type employees.employee.

To be more specific, you are binding the data for the DropDownList control to the *repeating* node of returned data.

When I save the file and run the application, you can see that the control displays the words Object Object in the data control.

This is because you are binding *all* the employee data to this one control and it doesn't know which field you actually want to display.

I am adding the labelField to the DropDownList control and then typing lastName for its value.

Now when I save the file and run the application, the last name for each employee is displayed.

Now, I want to populate the Office Phone field with the phone number of the employee selected in the DropDownList control.

I am adding a text property to the phone TextInput control.

For the value, I am adding the curly braces for the binding.

Remember that the DropDownList control has an id property named dropDownList with a lowercase d.

I am typing that name in the binding statement and then typing a period.

The employee information that I want to access is for the selected employee.

Luckily, the DropDownList control has a convenient selectedItem property that contains all the data stored for the selected item in the control.

I specifically want to access the phone field of the employee data.

When I save the file and run the application and then select an employee, you can see that the Office Phone number is populated with the selected employee's phone number.

For your next step, work through the exercise titled “Adding data to your application”.