

Using XML with E4X

Modify HTTPService to return E4X format

1. In **AdobeODT.mxml**, locate the **HTTPService** call.
2. Add the `resultFormat` property with a value of `e4x`.

```
<mx:HTTPService id="rooms"
    url="assets/roomList.xml"
    result="httpResultHandler(event)"
    fault="httpFaultHandler(event)"
    resultFormat="e4x" />
```

3. Locate the **Script** block and import the `XMLListCollection` class after the last import statement.

```
import mx.collections.XMLListCollection;
```

4. Change the **roomList** variable type to `XMLListCollection`;

```
private var roomList:XMLListCollection;
```

5. Create and instantiate a bindable private variable named `searchList` datatyped to `XMLListCollection`.

```
[Bindable] private var searchList: XMLListCollection = new
XMLListCollection();
```

6. Locate the **httpResultHandler** function and within the function as the first line, create and instantiate a locale variable named `xList` datatyped as `XMLList`. Pass `event.result` as `XML` as the parameter.

```
var xList:XMLList = new XMLList(event.result as XML);
```

7. Create a new `XMLListCollection` and use the `children()` method of `xList` as the parameter. Assign this to **roomList** as the new value.

```
roomList = new XMLListCollection(xList.children());
```

Modify component for XML data

8. Open **RoomRenderer.mxml** and locate the **Script** block.
9. Remove the **ArrayCollection** class.
10. Locate the **createlist** function and change the **eData** parameter type to `XMLList`.

```
private function createlist(eData:XMLList):String{  
    . . .
```

11. Within the function, remove the value of **a** and instantiate as a new `Array`.

```
var a:Array = new Array();
```

12. Using `for each`, use the `child()` method in `eData` to retrieve the event and set it as `sData` datatype as `String`.
13. Add the **sData** value to the **a Array** using the `push()` method.

```
for each (var sData:String in eData.child('event'))  
    a.push(sData);
```

Your code should appear as follows:

```
private function createlist(eData:XMLList):String{  
    var a:Array = new Array();  
    for each (var sData:String in eData.child('event'))  
        a.push(sData);  
  
    eventList = a.toString();  
    return eventList;  
}
```

14. Locate the last **Text** control.
15. Change the **text** value from **data.eventtypes.event** to `data.eventtypes`.

```
<mx:Text x="10" y="160"  
text="{createlist(data.eventtypes)}"/>
```

16. Save the file and run.

You should not see a difference in the appearance of your application.

Create search component

17. In **AdobeODT.mxml**, locate the **Label** control with the text **Rooms Available**.
18. Remove the **Label** control.
19. In the **Navigator** view, right click on the components folder and select **New > MXML Component**.
20. The filename is Search and is based on Form. Remove the **width** and **height** values.
21. Click **Finish**.
22. Within the **Form**, add `FormItem` tags with a `label` property having a value of Room Size For: and a `direction` property that has a value of horizontal.

```
<mx:FormItem label="Room Size For:" direction="horizontal">
</mx:FormItem>
```

23. Within the **FormItem** container add a `TextInput` control with an `id` property having a value of size, add a `restrict` property that has a value of 0-9 and add a `width` property that has a value of 75.

```
<mx:TextInput id="size" restrict="0-9" width="75"/>
```

24. Add a `Button` control with a `label` property having a value of Search and a `click` event that has a handler named `search()`.

```
<mx:Button label="Search" click="search()"/>
```

25. Add another `Button` control with a `label` property having a value Reset and a `click` event that has a handler named `reset()`.

```
<mx:Button label="Reset" click="reset()"/>
```

26. Save the file.

Create a custom event class

27. From the **events** folder open **OpenSelectEvent.as**.
28. Save it as `SearchEvent.as`.
29. In the file, change the class definition from **OpenSelectEvent** to `SearchEvent`.

```
public class SearchEvent extends Event
```

30. Change the **option** variable from **option** to `num` and change the datatype from **String** to `int`.

```
public var num:int;
```

31. Change the constructor parameter from **option:String** to `num:int`.

```
public function OptionSelectEvent(type:String, num:int )
```

32. Change the constructor name from **OptionSelectEvent** to `SearchEvent`.

33. Within the constructor change the **this.option = option** to `this.num = num`.

```
this.num = num;
```

34. Locate the **clone** function.

35. Within the function change the return event from **OptionSelectEvent** to `SearchEvent` and change the parameter from **option** to `num`.

```
return new SearchEvent(type, num);
```

36. Save the file.

Create event handlers for search form

37. Return to **Search.mxml**.

38. After the beginning **Form** tag, create a `Script` block.

39. Within the **Script** block import the `SearchEvent` class.

```
import events.SearchEvent;
```

40. Create a private function named `search` that takes no parameter and returns `void`.

41. Within the function create a variable named `searchObjEvent` datatyped as `SearchEvent`.

42. Assign to the `searchObjEvent` variable a new `SearchEvent` and pass two parameters: `searchEvent` and `size.text` cast as a `Number`.

43. Using the `dispatchEvent()` method `dispatch` `searchObjEvent`.

```
private function search():void{
```

```

        var searchObjEvent:SearchEvent = new
        SearchEvent("searchEvent", Number(size.text));

        dispatchEvent(searchObjEvent);
    }

```

44. Create a private function named `reset` that takes no parameter and returns `void`.
45. Within the function create a variable named `resetListEvent` datatyped as `Event`. To the `resetListEvent` variable assign a new `Event` and pass one parameter named `resetList`.
46. Using the `dispatchEvent()` method dispatch `resetListEvent`.

```

private function reset():void{
    var resetListEvent:Event = new Event("resetList");
    dispatchEvent(resetListEvent);
}

```

47. After the **Script** block, create Metadata tags.

```

<mx:Metadata>
</mx:Metadata>

```

48. Between the tags, create an `Event` with a name of `searchEvent` and a type of `events.SearchEvent`.

```

[Event(name="searchEvent", type="events.SearchEvent")]

```

49. Add another `Event` with a name of `resetList` and type of `flash.events.Event`.

```

[Event(name="resetList", type="flash.events.Event")]

```

50. Save the file.

Use the search component

51. Return to **AdobeODT.mxml** and locate the **ReservationForm** component.

52. Before the **ReservationForm** add the **Search** component from the **comp** namespace. Add the following properties:

- a. `x = 10`
- b. `y = 105`
- c. `searchEvent = search(event)`
- d. `resetList = resetList(event)`

Your code should appear as follows:

```
<comp:Search x="10" y="105"
  searchEvent="search(event)"
  resetList="resetList(event)"/>
```

53. Change the **List** `y` property to 162.

54. Locate the **Script** block and after the last import statement, import the **SearchEvent** class.

```
import events.SearchEvent;
```

55. Before the end of the **Script** block, create a private function named `search` that takes one parameter named `event` datatyped as `SearchEvent`. The function returns `void`.

56. Create and instantiate another local variable named `newXML` datatyped as `XML` and pass `<rooms></rooms>` as the parameter.

57. Create another local variable named `r1` datatyped as `XMLList` and has a value of `roomList.source`.

```
var newXML:XML = new XML('<rooms></rooms>');
var r1:XMLList = roomList.source;
```

58. Using `for each` loop over `r1` and set the XML value to `x` datatype as `XML`.

```
for each (var x:XML in r1){
}
}
```

59. Within the loop create a conditional. Use the `child()` method of `x` to get the `capacity` and evaluate to see if it is greater or equal to `event.num`.

60. If the conditional evaluates to true, use the `appendChild()` method of `newXML` and pass `x` as the parameter.

```

for each (var x:XML in r1){
    if (x.child('capacity') >= event.num){

        newXML.appendChild(x);
    }
}

```

61. After the loop, use the `children()` method of `newXML` as the parameter to instantiate a new `XMLList` object. Assign it to `searchList.source`.

```
searchList.source = new XMLList(newXML.children());
```

62. Assign `searchList.source` to `dg.dataProvider`.

```
dg.dataProvider = searchList.source;
```

63. Before the end of the **Script** block, create a private function named `resetList` that takes one parameter named `event` datatyped as `Event`. The function returns `void`.

64. Within the function assign the `roomList.source` to `dg.dataProvider`.

```
private function resetList(event:Event):void{
    dg.dataProvider = roomList.source;
}

```

65. Save the file and run.

66. Enter a 150 into the search field and click the **Search** button.

You should see all rooms that have a capacity of 150 and higher.

67. Click the **Reset** button.

You should see all the rooms again.