Providing XML to Controls with E4X

In this section of the course, You will learn how to use XML objects in your applications using E4X. You will also learn how to use these XML objects as data providers, using the XMLListCollection class.

Objectives

After completing this unit, you should be able to:

- Understand the structure of an XML object
- Use E4X to find data in the XML object
- Populate the Tree control with XML data
- Use various XML classes, such as XML, XMLList and XMLListCollection
- Use various methods and properties of the XML classes
- Add and remove items from an XML based shopping cart

Topics

In this unit, you will learn about the following topics:

- Understanding XML structure
- Working with XML data in ActionScript 3
- Using the Tree control
- Using the ICollectionView Interface

Acknowledgements

► This training is one chapter of the authorized training available from Adobe. It is provided for you by the Adobe Partner Enablement team. For the complete list of available hands-on Flex training courses please see:

http://www.adobe.com/training/instructor_led/

► For authorized training centers please see:

http://partners.adobe.com/public/partnerfinder/tp/show_find.do

► Thanks also the Adobe User Group team for distribution.

http://www.adobe.com/cfusion/usergroups/

► See more chapters of the authorized training on the Developer Center.

http://www.adobe.com/devnet/

Understanding XML structure

- XML is a standard way to describe data objects
- XML is a way to represent complex data in text format
- Most development platforms and applications have an XML parser
  - Because of that XML is often used to share (complex) data between software systems
- Can be seen as the 'lingua franca' for data; many (programming) languages understand it

General syntax

- At its simplest an XML document is a set of containers called elements (also known as nodes)
- Tags used to mark the boundaries between elements
- The root element contains everything
  - In the example below the root element is **<products>**
- Sub-elements or child elements are contained in the root element
  - In the example below there are two **<product>** sub-elements
- Attributes are properties which modify a given element and are specified by using an attribute/value pair
  - In the example below the **<product>** tags have an **id** attribute and a corresponding value

```
var myXML:XML =
<products>
   <product id="1">
     <name>Broccoli</name>
     <description lang="en">Firm and no
   bitterness</description>
     <description lang="nl">Stevig en niet
   bitter</description>
   </product>
   <product id="2">
     <name>Bananas</name>
     <description lang="en">Bunches of Bananas,
   fresh off the tree</description>
     <description lang="nl">Bananen per tros,
   vers geplukt</description>
   </product>
</products>
```

Working with XML data in ActionScript 3

- ActionScript 3 incorporates XML as a native data structure
- ActionScript 3 has a new way of working with XML
  - Ways of working with XML in prior versions of ActionScript not based on ECMAScript standards
- This is based on ECMAScript Edition 4
- New classes and functionality collectively called E4X (ECMAScript for XML)

E4X introduction

- E4X (ECMAScript for XML) is a new set of classes and functionality for working with XML data in ECMAScript.
- More information can be found at http://www.ecma-international.org/publications/standards/ECMA-357.htm
- ActionScript 3 includes this new standard and implements the new E4X classes: XML, XMLList, XMLListCollection, QName and Namespace
- These classes make working with XML data in ActionScript 3:
  - Simpler - no complex loops necessary
  - More consistent - both internally and with other parts of ActionScript
  - More familiar - uses familiar syntax

Note: The XML Class from ActionScript 2.0 has been renamed XMLDocument in ActionScript 3 and is included primarily for legacy support.

E4X classes

- XML and XMLList classes implement E4X XML handling standards
  - XML: Contains methods and properties for working with XML objects
  - XMLList: Is an ordered collection of XML (can be more than one XML object)

XML

- Example properties
  - **`ignoreComments`**: Determines whether XML comments are ignored when XML objects parse the source XML data
  - **`ingoreWhitespace`**: Determines whether white space characters at the beginning and end of text nodes are ignored during parsing

- Example methods
  - **`appendChild()`**: Appends the given child to the end of the XML object's properties
  - **`childIndex()`**: Identifies the zero-indexed position of this XML object within the context of its parent
  - **`children()`**: Lists the children of the XML object in the sequence in which they appear

XMLList

- An XMLList object is an ordered collection of properties.
  - Represents an XML document, an XML fragment, or an arbitrary collection of XML objects.
- An individual XML object is the same thing as an XMLList containing only that XML object.
- All methods of the XML class are also available for an XMLList object that contains exactly one XML object, otherwise some methods not allowed for XMLList
  - Example: **`childIndex()`** does not make sense on an XMLList
- Differences with XML class
  - The **`length()`** method of XML class is always 1, whereas the **`length()`** method of XMLList returns the number of properties in the XMLList object

E4X operators

- Used to access attributes and elements of and XML object
- Can be used for both data retrieval and assignment

► The following XML object will be used for operator examples

```
mealsXML=
<meals>
  <serving time="breakfast">
    <lite cost="5.95">
      <meat>spam</meat>
      <waffle>plain</waffle>
    </lite>
    <hearty cost="9.95">
      <meat>bacon and sausage</meat>
      <waffle>blueberry</waffle>
    </hearty>
  </serving>
  <serving time="lunch">
    <lite cost="10.95">
      <meat>turkey</meat>
      <sandwich>mustard only</sandwich>
    </lite>
    <hearty cost="12.95">
      <meat>pastrami</meat>
      <sandwich>triple-decker</sandwich>
    </hearty>
  </serving>
  <serving time="dinner">
    <lite cost="11.95">
      <meat>chicken breast</meat>
      <side>steamed veggies</side>
    </lite>
    <hearty cost="14.95">
      <meat>t-bone</meat>
      <side>garlic mashed potatos</side>
    </hearty>
  </serving>
</meals>
```

Dot (.) operator

- Use the XML **dot** (.) operator to navigate to child elements of an XML object

```
mealsXML.serving[1]
<serving time="lunch">
  <lite cost="10.95">
    <meat>turkey</meat>
    <sandwich>mustard only</sandwich>
  </lite>
  <hearty cost="12.95">
    <meat>pastrami</meat>
    <sandwich>triple-decker</sandwich>
  </hearty>
</serving>
----------------------------
mealsXML.serving[2].hearty.meat
t-bone
```

Parentheses [( )] operators

- Named **Predicate Filtering**
- Use the (and) operators to evaluate an expression in an E4X XML construct
- Use parentheses to insert conditional search logic with the dot operator
  - Can also use Regul**ar expressions**

```
mealsXML.serving.lite.(meat=='turkey')
<lite cost="5.95">
  <meat>turkey</meat>
  <waffle>plain</waffle>
</lite>
<lite cost="7.95">
  <meat>turkey</meat>
  <sandwich>mustard only</sandwich>
</lite>
```

Attribute (@) operator

- Use the XML **attribute** (@) operator to identify attributes of an XML object
  - Use in combination with the dot operator and parentheses for searches

```
mealsXML.serving.hearty.(@cost == 9.95)
<hearty cost="9.95">
  <meat>bacon and sausage</meat>
  <waffle>blueberry</waffle>
</hearty>
<hearty cost="9.95">
  <meat>pastrami</meat>
  <sandwich>triple-decker</sandwich>
</hearty>
```

- Can use operators multiple times

```
mealsXML.serving.hearty.(@cost ==
   9.95).(meat=='pastrami')
<hearty cost="9.95">
  <meat>pastrami</meat>
  <sandwich>triple-decker</sandwich>
</hearty>
```

Descendent accessor (..) operator

- Use the XML **descendent accessor** (..) operator to navigate to descendant elements of an XML object

```
mealsXML..lite
<lite cost="5.95">
  <meat>turkey</meat>
  <waffle>plain</waffle>
</lite>
<lite cost="7.95">
  <meat>turkey</meat>
  <sandwich>mustard only</sandwich>
</lite>
<lite cost="12.95">
  <meat>chicken breast</meat>
  <side>steamed veggies</side>
</lite>
```

- Use in combination with the dot and attribute operators and parentheses for searches

```
mealsXML..lite.(@cost > 6)
<lite cost="7.95">
  <meat>turkey</meat>
  <sandwich>mustard only</sandwich>
</lite>
<lite cost="12.95">
  <meat>chicken breast</meat>
  <side>steamed veggies</side>
</lite>
```

Data assignment

► With dot and attribute operators can also assign data:

```
mealsXML.serving[0].lite.@cost='100'
mealsXML.serving[0].lite.meat='turkey bacon'
<serving time="breakfast">
  <lite cost="100">
    <meat>turkey bacon</meat>
    <waffle>plain</waffle>
  </lite>
  <hearty cost="9.95">
    <meat>bacon and sausage</meat>
    <waffle>blueberry</waffle>
  </hearty>
</serving>
```

Using the Tree control

- The Tree control displays hierarchical data as an expandable tree
- A Tree control consists of branches and leaf nodes. An item in a Tree is either a branch or a leaf
- Branch and leaf have default icons, a folder icon for a branch and a file icon for a leaf. Tree icons can be changed by specifying the **`folderOpenIcon`**, **`folderClosedIcon`** and **`defaultLeafIcon`** properties of the Tree control.

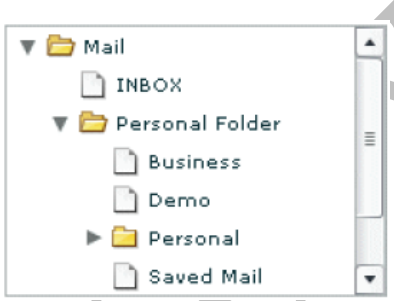

Figure 1: The Tree control

Creating a Tree

```
<mx:Tree dataProvider="{dp}">
```

- When you assign a dataProvider, the Tree class handles the source data object as follows:
  - A String containing valid XML text is converted to an XML object
  - An XMLNode is converted to an XML object
  - An XMLList is converted to an XMLListCollection
  - Any object that implements the ICollectionView interface is cast to an ICollectionView
  - An Array is converted to an ArrayCollection
  - Any other type object is wrapped in an Array with the object as its sole entry
- As with all data provider controls, if the data provider content can change outside the control and an automatic screen update is required, wrap the the data source in an ArrayCollection or an XMLListCollection.
- The Tree control reads the XML data and builds the Tree based on the nested relationship of the data in the data provider.
- Specify the name of the property or attribute to be used as the label with the **`labelField`** property of the Tree control

Tree Nodes

- Nodes at the highest level are called root nodes
  - A Tree can have more than one root node
  - To prevent the displaying of a root node, specify the **`showRoot`** property of the Tree control false.
- Branch Nodes
  - A Branch Node can contain multiple Child Nodes
  - Child Nodes can be branches or leafs
- Leaf Nodes
  - A Leaf Node is an endpoint in the Tree
  - The following code is used to display the Tree in figure on previous page

```
<mx:Tree id="tree1"
 labelField="@label"
 width="20%">
 <mx:dataProvider>
  <mx:XML format="e4x">
   <folder label="Mail">
    <folder label="INBOX"/>
    <folder label="Personal Folder">
     <Pfolder label="Business" />
     <Pfolder label="Demo" />
     <Pfolder label="Personal"
      isBranch="true" />
     <Pfolder label="Saved Mail" />
    </folder>
    <folder label="Sent" />
    <folder label="Trash" />
   </folder>
  </mx:XML>
 </mx:dataProvider>
</mx:Tree>
```

Walkthrough 1: Retrieving XML data and populating a Tree

In this walkthrough, you will perform the following tasks:

- Create an event handler for an HTTPService call that retrieves XML from a server
- Use the XML data to populate a Tree

Steps

1. Close any open projects and then open the **E4X** project.
2. Open **E4X_wt1.mxml**.
3. Run the application.

   You should see the Cafe Townsend Delivery Service application with 3 panels. In the left panel you need to display the dishes in a Tree.

Returning E4X data

4. In the HTTPService tag just below the Script block, add a **`result`** event and call a function named ***`dishesXMLHandler()`***. Pass the **`event`** object as a parameter.
5. In the HTTPService tag, use the **`resultFormat`** property to specify you want the XML object to be deserialized so that you can access the data using ECMAScript for XML (e4x).

```
<mx:HTTPService
  id="dishesXML"
  url="assets/xml/dishes.xml"
  showBusyCursor="true"
  result="dishesXMLHandler(event)"
  resultFormat="e4x" />
```

6. Inside the Script block, place a **`trace()`** statement inside the ***`dishesXMLHandler()`*** event handler and display the **`result`** property of the **`event`** object.

```
private function
   dishesXMLHandler(event:ResultEvent):void
{
   trace(event.result);
}
```

7. Debug the application.

   You should see the XML object being displayed in your Console view.

Tip: *Double click the Console view tab to see it full screen.*

8. Terminate the debugging session.

Populating a Tree

9. In the Script block just below the import, define a private, bindable variable named ***dishesDP*** datatyped as **XMLList**.

```
[Bindable]
private var dishesDP:XMLList;
```

10. Inside the ***dishesXMLHandler()*** event handler, remove the **trace()** method and populate the ***dishesDP*** variable with all **category** nodes and their child nodes. Use the '..' operator.

```
dishesDP = event.result..category;
```

11. Inside the left Panel of the application, insert a Tree tag. Add the following properties:
    - id: dishesTree
    - dataProvider: {dishesDP}
    - labelField: @name
    - width: 100%
    - height: 100%
    - borderThickness:0

```
<mx:Tree id="dishesTree"
    dataProvider="{dishesDP}"
    labelField="@name"
    width="100%"
    height="100%"
    borderThickness="0" />
```

12. Run the application.

    You should see the populated Tree in the left panel. Be sure you can open the branch nodes and see the actual dishes.

Tree properties

- Tree has number of properties
  - Previously used **`dataProvider`**
- The **`selectedItem`** property holds the selected node from a Tree
  - Also **`selectedItems`** if allowing multiple selections
  - Selected node will usually be an XML object
- Other helpful properties for controlling display
  - **`showRoot`**: Boolean: Sets the visibility of the top level nodes
  - **`firstVisibleItem`**: The node that is currently displayed in the top row of the tree

Walkthrough 2: Displaying the selected item in a form

In this walkthrough, you will perform the following tasks:

- Use the XML data from the Tree to populate a form that is a detailed view of a selected dish.

Steps

1. Open E4X_wt2.mxml.

Populating the form

2. Add a **`change`** event to the Tree. In the ActionScript call a function named ***`populateForm()`*** and send the **`event`** object as a parameter.

```
change="populateForm(event)"
```

3. Locate the private function named ***`populateForm()`***.
4. In the function, remove the **null** value that is assigned to the ***`node`*** variable. Assign ***`node`*** as a pointer to the selected XML-node of the Tree. Use the **`target`** property of the event object.

```
var node:XML = event.target.selectedItem;
```

Note: Only selected dishes, not categories, should be displayed in the form. To do this an **`if-else`** *statement is used that checks for the existence of a* ***price*** *attribute using the '@' operator inside the node.*

5. Run the application.

You should see that the form will be populated and an image will displayed, when you select a dish from the Tree. When you select a category in the Tree, the form should become invisible again.

Using XML objects

- You may want to use XML data in a way other than as a dataProvider for a Tree
- Can display XML data using String conversion functions
- Use XML data as a dataProvider to list-based controls using XMLListCollection

XML Type Conversion

- Converting XML objects to Strings
  - To convert an XML object to a String, use the **`toString()`** method
  - The **`toString()`** method returns a string with all namespace declarations, tags, attributes and content from the XML object.
- Converting Strings to XML objects
  - Instantiate the XML Class to create an XML object.
  - Specify a String as an argument

```
var myXML:XML = new XML(<products></products>)
```

XMLListCollection

- The XMLListCollection class provides collection functionality to an XMLList object
- An XMLListCollection object is used as a data provider to enable automatic data provider updates when the underlying XML data changes.
- Must import the class to use it

```
mx.collections.XMLListCollection
```

- Placing an XMLList or XML object into an **`XMLListCollection`** class has the following advantages
  - The elements of the XMLListCollection can be used in bindings that will continue to be monitored
    - This is not the case with the normal **`XMLList`** class, once an element from an XMLLIst or XML object is used in a binding it is no longer monitored
  - Implements both the ICollectionView and IList interfaces
    - Provides rich set of tools for data manipulation
- Recommended data structure when data used as a data provider
- Analogous to using an ArrayCollection instead of the Array, you would use an XMLListCollection instead of an XMLList

Tip: *Can use the filtering options of the ICollectionView, but often the E4X methods are more powerful and easier to implement.*

Walkthrough 3: Adding items to the shopping cart

In this walkthrough, you will perform the following tasks:

- Create an XML based shopping cart
- Add items to the shopping cart
- Add the same item to the shopping cart and update the quantity

Steps

1. Open E4X_wt3.mxml.

Creating the shopping cart

2. Create a bindable, private variable named ***ShoppingCartXML*** datatyped as **XML**. Populate the new variable with an instance of the XML class with a root node of ***<products></products>***.

```
[Bindable]
private var shoppingCartXML:XML = new
   XML(<products></products>);
```

3. Import the **mx.collections.XMLListCollection** class.
4. Create a bindable, private variable named ***ShoppingCartData*** datatyped as **XMLListCollection**.

```
[Bindable]
private var
   shoppingCartData:XMLListCollection;
```

Adding an item to the shopping cart

5. Locate the function named ***addToShoppingCart()***.
6. Inside the function, create a variable local to the function named ***newItem*** to hold the product to add to the shopping cart.

*Note: To save on typing with no learning point, copy the code in the **wt_3TypingHelp.txt** file and paste it into the function.*

7. Append the new shopping cart item to the shoppingCart XML object.

```
shoppingCartXML.appendChild(newItem);
```

8. Set the ***shoppingCartData*** equal to a new **XMLListCollection** and use **shoppingCartXML.children()** method to populate it.

```
shoppingCartData = new XMLListCollection
   (shoppingCartXML.children());
```

Displaying the shopping cart in the DataGrid

9. Bind the ***shoppingCartData*** XMLListCollection as the **dataProvider** for the DataGrid in the right Panel.
10. Run the application.

    You should see the products in the Tree and you should be able to add them to the shopping cart. Note that items are added as separate items when you add the same item multiple times.

Updating an item already in the shopping cart

11. Locate the ***addToShoppingCart()*** function.
12. At the top of the function, create a variable local to the function named ***itemToUpdate*** datatyped as an **XMLList**. Use the E4X attribute operator (**@**) and the **==** comparison operator to populate it with selected node from the Tree. Use the ***id*** attribute in the comparison.

```
var itemToUpdate:XMLList =
   shoppingCartXML.product.(@id==
   dishesTree.selectedItem.@id);
```

Note: If the item is already in the shopping cart, itemToUpdate will hold a pointer to the node in shoppingCartXML.

13. Insert an **if** statement that checks if the **length** of the ***itemToUpdate*** variable is greater than 0. If it is, add 1 to the ***qty*** property of the variable. If it is not, add the new item to the shopping cart using the existing logic.

Tip: *Since the itemToUpdate is a reference to the node in the shoppingCartXML, you can use* ***itemToUpdate.qty*** *to update the quantity. The easiest way to add 1 to the value is to use the postfix increments operator:* ***itemToUpdate.qty++****.*

Note: Make sure the line that populates the shoppingCartData with a new instance of XMLListCollection is outside the if statement.

14. Check to be sure your function appears as follows:

```
private function addToShoppingCart():void
{
 var itemToUpdate:XMLList =
  shoppingCartXML.product.(@id ==
  dishesTree.selectedItem.@id);
 if(itemToUpdate.length() > 0)
 {
  itemToUpdate.qty++;
 }
 else
 {
  var newItem:XML =
  <product id={dishesTree.selectedItem.@id}>
   <name>
    {dishesTree.selectedItem.@name}
   </name>
   <price>
    {dishesTree.selectedItem.@price}
   </price>
   <qty>
    1
   </qty>
  </product>;
  shoppingCartXML.appendChild(newItem);
 }
 shoppingCartData = new
  XMLListCollection(shoppingCartXML.children(
  ));
}
```

15. Run the application.

    You should see that when you add the same product multiple times to the shopping cart the quantity is increased.

Removing XML items from a list

- When removing items from a list you have two options
  - Remove from the XMLList
  - Remove from the XMLListCollection
- Since the XMLListCollection is nothing more than a view onto the List, both accomplish the same thing

Using an XMLListCollection

- Use the IList methods (implemented in XMLListCollection)

```
instance.removeItemAt()
instance.removeAll()
```

Using the XML delete operator

- Can also remove attributes or elements using the XML **`delete`** operator
- General syntax

```
delete reference
```

  - The reference is and XMLList object
- Example

Table 1: Example of XML delete operator

Code	Result
`var x1:XML = <x1> <a id="2">AYY</a> <a>AYY 2 </a> <b>BEE</b> <c>CEE</c> </x1>;`	`<x1> <a id="52">AYY</a> <a>AYY 2</a> <b>BEE</b> <c>CEE</c> </x1>`
`delete x1.a.@id;`	`<x1> <a>AYY</a> <a>AYY 2</a> <b>BEE</b> <c>CEE</c> </x1>`
`delete x1.b;`	`<x1> <a>AYY</a> <a>AYY 2</a> <c>CEE</c> </x1>`
`delete x1.a;`	`<x1> <c>CEE</c> </x1>`

Walkthrough 4: Removing items from and clearing the cart

In this walkthrough, you will perform the following tasks:

- Create functionality to remove an item from the shopping cart
- Create functionality to clear the shopping cart

Steps

1. Open E4X_wt4.mxml.
2. Run the application.

*Note: Note that the right panel has two buttons, one labeled **Remove** the other **Clear Cart**. Both are initially disabled.*

Enabling the Remove and Clear Buttons

3. Locate the ***removeButton*** Button instance. Change the **enabled** property from **false** to a binding that checks to see if the **selectedIndex** of the ***shoppingCart*** DataGrid is not equal to ***-1***.

```
enabled="{shoppingCart.selectedIndex != -1}"
```

Note: A selectedIndex has the value of -1 when nothing is selected.

4. Locate the ***clearButton*** Button instance. Change the **enabled** property from **false** to the **length** of the ***shoppingCartData*** variable. Cast the value as a Boolean.

```
enabled="{Boolean(shoppingCartData.length)}"
```

Note: You must cast the numeric value of length to a Boolean since the enabled property expects a Boolean value. The only value interpreted as false will 0, and that is the only time you want the Clear Button disabled.

Removing a single item from the cart

5. Locate the ***removeFromShoppingCart()*** method.
6. Inside the function, create a variable local to the function named ***itemToRemove*** datatyped as **Object**.

7. Set the new variable equal to the **selectedItem** from the ***shoppingCart***.

```
var itemToRemove:Object=
   shoppingCart.selectedItem;
```

8. Delete the node found from the ***shoppingCartData*** using the **removeItemAt()** method of the XMLListCollection. Since the **removeItemAt()** method requires the ordinal position, use the **childIndex()** method to retrieve that.

```
shoppingCartData.removeItemAt(itemToRemove.
childIndex());
```

Note: This could also be done using the delete operator and the shoppingCartXML XMLList: delete shoppingCartXML.product[itemToRemove.childIndex()];

9. Your code should look like this:

```
private function removeFromShoppingCart():void
{
  var itemToRemove:Object=
    shoppingCart.selectedItem;
  shoppingCartData.
    removeItemAt(itemToRemove.childIndex());
}
```

10. Run the application.

    You should see that when you add items to the shopping cart, then select an item you can delete it.

Clearing the cart

11. Locate the ***clearCart()*** function.
12. Inside the function clear the shopping cart using the **removeAll()** method of the ***shoppingCartData*** XMLListCollection.

Note: Since this is a reference to the shoppingCartXML XMLList, the underlying XML will be cleared also.

13. Your code should appear as follows:

```
private function clearCart():void
{
  shoppingCartData.removeAll();
}
```

14. Run the application.

    You should be able to remove items from the cart and clear the shopping cart.

Summary

- XML is a native data structure in ActionScript 3
- ECMAScript for XML (E4X) defines how XML handling is implemented in AS3
  - E4X defines a set of classes and functionality for working with XML
- XML class has the following methods that are often useful
  - **`appendChild()`**: Appends the given child to the end of the XML object's properties
  - **`childIndex()`**: Identifies the zero-indexed position of this XML object within the context of its parent
  - **`children()`**: Lists the children of the XML object in the sequence in which they appear
- E4X has a rich set of operators to use on XML including
  - Dot (.): Navigate to child elements
  - Parentheses: Evaluate expressions (also called predicate filtering)
  - @: Identify attributes
  - Descendent accessor (..): Navigate to descendant elements
  - Data assignment: Assign data with dot and attribute operators
- Tree control displays hierarchical data as an expandable tree
  - Consumes XML data as dataProvider
- Levels of Tree are: root nodes, branch nodes and leaf nodes
- When a node of a Tree is selected that item is stored in the **`selectedItem`** property
- XMLListCollection has the same advantages the ArrayCollection has since it implements the same interfaces
- Remove XML data from an XML object using
  - From an XMLListCollection

```
instance.removeItemAt()
instance.removeAll()
```

  - From XML

```
delete reference
```

Review

1. What is E4X?

2. Name three methods of the XML class and what they do.

3. Name five E4X operators and what they do.

4. Name the different nodes of a Tree

5. Name three properties of the Tree class and what they represent

6. Name three ways to remove data from an XML object.

Set up

1. Unzip the file ***e4xStudentFiles.zip*** to the root of the C: drive.

Note: The zip will install files to the C:\adobeTraining\flex3 directory.

2. In Flex Builder 3 select ***File > Switch Workspace ...***
3. Select the ***C:\adobeTraining\flex3\e4xStudentFiles_workspace*** folder.