

Flex 3 and ActionScript 3.0 provide *ECMAScript for XML*, or *E4X*, implementation that enables you to work with dot notation to access the nodes within an XML hierarchy. E4X provides a simplified shorthand notation so you can access specific nodes or sets of nodes by name or by index without complicated custom recursive functions. It also defines methods and properties for accessing all parts of the XML object, including comments, namespaces, and processing instructions. The Flex Framework and ActionScript 3.0 define two low-level objects for working with XML: the `XML` object and the `XMLList` object. The `XML` object represents a single XML element, an XML document with multiple children, or a single-value element within a document. The `XMLList` object represents a set of XML elements that are siblings of one another. `XMLList` objects do not require a top-level node, for example:

```
<item id="2" name="Chewing Gum"/>
<item id="3" name="Cotton Candy"/>
<item id="4" name="Candy Bar"/>
```

XML objects do require a top-level node:

```
<order>
  <item id="2" name="Chewing Gum"/>
  <item id="3" name="Cotton Candy"/>
  <item id="4" name="Candy Bar"/>
</order>
```

To perform XML transformations or to build XML documents, both the `XML` and `XMLList` objects define methods to append, rename, and re-parent nodes within an XML document. To perform queries, access particular attributes, or filter XML documents, use the E4X querying syntax. For working with XML namespaces, the `XMLUtil` class defines methods of accessing and comparing namespaces.

19.1 Load an XML File

Problem

You need to load an external XML file and process it.

Solution

Use the `HTTPService` component to load an XML file and set `resultFormat` to `xml`. Alternatively, use the `flash.net.URLLoader` class to load XML files by creating a `URLLoader` instance and calling the `load` method.

Discussion

By default, the `HTTPService` component converts any loaded XML into an ActionScript object. To ensure that the `HTTPService` component does not convert the loaded XML to an object, set the `resultFormat` property of the `HTTPService` object to `E4X` as shown:

```
<mx:HTTPService url="http://server/xmlDoc.xml" id="xmlService" resultFormat="e4x"
result="xmlObj = XML(xmlService.lastResult)"/>
```

The `lastResult` of the `HTTPService` is used to set the value of the `xmlObj` variable here.

If the XML file is static or you need to load it only once, it is easier and lighter to use the `flash.net.URLLoader` class to load the data:

```
private var loader:URLLoader = new URLLoader();

private function init():void {
    loader.addEventListener(Event.COMPLETE, setResult);
    var req:URLRequest = new URLRequest();
    req.url = "http://server/xmlDoc.xml";
    loader.load(req);
}

private function setResult(event:Event):void {
    trace(XML(loader.data).toString());
}
```

The `Event.COMPLETE` event listener must be added to the `URLLoader` so that the application can be notified when the file is finished being loaded. The loaded data is stored as the `URLLoader.data` property and is not cast or altered at all. If the XML data will be dynamically generated or will need to be loaded multiple times, it may be easier to load the file by using `HTTPService`. A complete code listing using both methods is shown here:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300"
creationComplete="init()">
<!-- if the resultFormat is not set, then the result will be parsed as an Object -->
<mx:HTTPService url="http://server/xmlDoc.xml" id="xmlService" resultFormat="e4x"
result="xmlObj = XML(xmlService.lastResult)"/>

<mx:Script>
```

```

<![CDATA[
    private var loader:URLLoader = new URLLoader();

    [Bindable]
    private var xmlObj:XML;

    private function init():void {
        loader.addEventListener(Event.COMPLETE, setResult);
    }

    private function urlLoaderSend():void {
        var req:URLRequest = new URLRequest();
        req.url = "http://server/xmlDoc.xml";
        loader.load(req);
    }

    private function setResult(event:Event):void {
        xmlObj = (loader.data as XML)
    }

    ]]>
</mx:Script>
<mx:Button click="xmlService.send()"/>
<mx:Button click="urlLoaderSend()" label="load via URLLoader"/>
<mx:TextArea text="{xmlObj.toString()}" />
</mx:VBox>

```

19.2 Navigate an XML Document in E4X

Problem

You need to select certain nodes from an XML file based on properties of the attributes.

Solution

Use the E4X syntax's @ operator to access attributes, the ([]) operator (indexed array) to indicate multiple child relationships, and the . operator to indicate named child relationships.

Discussion

With E4X, you can navigate an XML file to access specific children by using the . operator after the child's name. For example, from this file

```

var xml:XML = <foo>
  <bar>Hello World</bar>
</foo>

```

you could access the value of <bar> as shown:

```
xml.bar
```

Reference to the `<foo>` node is not needed because it is the root node of the XML object.

To access the attribute of a node from a file such as

```
var xml:XML = <foo>
<bar type="salutation">Hello World</bar>
</foo>
```

use the `@` operator to indicate that the desired property is an attribute:

```
xml.bar.@type
```

To access multiple child nodes of the same name, use the `[]` operator. From an example such as

```
var xml:XML = <foo>
<bar type="salutation">Hello World</bar>
<bar type="salutation">Hola</bar>
<bar type="salutation">Guten Tag</bar>
</foo>
```

you can access the third object in the series of `<bar>` like so:

```
xml.bar[2].@type
```

For a simple XML structure defining items on a menu, use a snippet like the following:

```
private var xmlItems:XML = <order>
    <item id='1'>
        <menuName>burger</menuName>
        <price>3.95</price>
    </item>
    <item id='2'>
        <menuName>fries</menuName>
        <price>1.45</price>
    </item>
</order>

private var arr:Array;

private function init():void {
    arr = new Array();
    for each(var xml:XML in xmlItems.item) {
        arr.push(Number(xml.@id));
    }
}
```

To test the value of an attribute or a node, use the equality operator (`==`):

```
trace(xmlItems.item.@id == "2").menuName);
```

Any node that meets all the criteria will be traced, and all others will be ignored. The following example will set the text of the `Label` component to the `menuName` of the `item` that has an `id` of 2:

```
private var xmlItems:XML = <order>
    <item id="1">
```

```

        <menuName>burger</menuName>
        <price>3.95</price>
    </item>
    <item id="2">
        <menuName>fries</menuName>
        <price>1.45</price>
    </item>
</order>

private function init():void {
    xmlLabel.text = xmlItems.item.@id == "2".menuName;
}

]]>
</mx:Script>
<mx:Label id="xmlLabel"/>

```

Both the equality (==) and the inequality (!=) operators can be used to test the value of an attribute or a node, as can any string or number method that returns a Boolean value.

19.3 Use Regular Expressions in E4X Queries

Problem

You need to create complex E4X queries by using regular expressions as part of the query.

Solution

Add the regular expression as a literal to the E4X statement and call the `test` method of the regular expression.

Discussion

When combined, regular expressions and E4X allow very precise filtering of XML nodes. The literal syntax of the regular expression enables you to add a regular expression without a call to a constructor and to use the `test` method of the regular expression on the attribute or value of the XML node. This line, for example, tests the `id` attribute of the `item` node:

```
xmlItems.item.(/\d\d\d/.test(@id)).price
```

Any items that possess a three-digit `id` property will have the price value of the property returned. Any item that does not either possess all of these fields or return true from the regular expression's `test` method will not have its value returned. The following code snippet shows an example with the XML looped through and the nodes each tested against the E4X expression:

```
private var xmlItems:XML =
    <order>
        <item id="1">
            <menuName>burger</menuName>

```

```

        <price>3.95</price>
    </item>
    <item id="100">
        <menuName>burger</menuName>
        <price>3.95</price>
    </item>
    <item id="2000">
        <menuName>fries</menuName>
        <price>1.45</price>
    </item>
</order>

```

```

private var arr:Array;

private function init():void {
    arr = new Array();
    for each ( var xml:XML in xmlItems) {
        arr.push(xmlItems.item(/^\d\d\d/.test(@id)).price);
    }
    trace(arr);
}

```

It is worth noting as well that you can use these E4X queries as a property for data binding within controls.

19.4 Add an XMLList to an XML Object

Problem

You need to append the value of an `XMLList` object to a node within an `XML` object.

Solution

Use an E4X expression to locate the desired node to which the `XMLList` object should be appended and then call the `appendChild` method on that node.

Discussion

You can add an `XMLList` directly to an `XML` object or to another `XMLList` object by using the `appendChild` method of the `XML` class. For example, say you are given this `XML` object:

```

var xmlNode:XML = <data>
    <item id="1"/>
    <item id="2"/>
    <item id="3"/>
</data>

```

Then, calling `appendChild` with a new node

```

var newXML:XML = <item id="4"/>
xmlNode.appendChild(newXML);

```

produces the following:

```

var xmlNode:XML = <data>
    <item id="1"/>
    <item id="2"/>
    <item id="3"/>
    <item id="4"/>
</data>

```

The new node will be added to the root node of the XML object. You can call the `appendChild` method on any node within the XML object as well:

```

    var list:XMLList = new XMLList('<characteristic name="cuts through metal"/>
<characteristic name="never dulls"/><characteristic name="dishwasher safe"/><characteristic
name="composite handle"/>');
    var node:XMLList = xmlNode.item.@id == 3);
    node.appendChild(list);
}

```

To add items from one `XMLList` to another, you could loop through the origin list and assign the indexed value of the origin to the destination `XMLList`:

```

var newXML:XMLList = new XMLList();
for(var i:int = 0; i<list.length(); i++) {
    newXML[i] = list[i];
}

```

This approach will not work, however, if `newXML` is of type `XML`. To loop through a list and add certain or all items from a list, use the `appendChild` method:

```

var newXML:XML = <data></data>;
for(var i:int = 0; i<list.length(); i++) {
    newXML.appendChild(list[i]);
}

```

19.5 Bind to an XMLList or an E4X Query

Problem

You want to bind a control to a value returned from an E4X query into an XML object.

Solution

Use the binding tags (`{}`) to wrap the E4X expression and set the property of the control.

Discussion

Say you are given the following XML file:

```

[Bindable]
private var xmlItems:XML =
    <CATALOG>
        <PLANT id="2">
            <COMMON>Bloodroot</COMMON>
            <BOTANICAL>Sanguinaria canadensis</BOTANICAL>
        </PLANT>
    </CATALOG>

```

```

        <ZONE>4</ZONE>
        <LIGHT>Mostly Shady</LIGHT>
        <PRICE>$2.44</PRICE>
        <AVAILABILITY>031599</AVAILABILITY>
    </PLANT>
    <PLANT id="3">
        <COMMON>Columbine</COMMON>
        <BOTANICAL>Aquilegia canadensis</BOTANICAL>
        <ZONE>3</ZONE>
        <LIGHT>Mostly Shady</LIGHT>
        <PRICE>$9.37</PRICE>
        <AVAILABILITY>030699</AVAILABILITY>
    </PLANT>
    <PLANT id="5">
        <COMMON>Marsh Marigold</COMMON>
        <BOTANICAL>Caltha palustris</BOTANICAL>
        <ZONE>4</ZONE>
        <LIGHT>Mostly Sunny</LIGHT>
        <PRICE>$6.81</PRICE>
        <AVAILABILITY>051799</AVAILABILITY>
    </PLANT>
</CATALOG>

```

The text value of a `Label` control can be set to the result of an E4X query that will return the `PRICE` of the `PLANT` where the `id` is 5:

```
<mx:Label text="{xmlItems.PLANT.(@id == 5).PRICE}"/>
```

Because the result of an E4X expression that returns multiple nodes is an `XMLList`, the value can be used to set the `dataProvider` for a `ComboBox`.

```
<mx:ComboBox dataProvider="{xmlItems.PLANT.(ZONE == 4).PRICE}"/>
```

Similarly, when an E4X expression returns multiple nodes and there is no subnode set in the E4X expression, the entire node is returned and can be used to set the `dataProvider` for a `DataGrid` control as shown:

```

<mx>DataGrid dataProvider="{xmlItems.PLANT.(ZONE == 4)}">
    <mx:columns>
        <mx:DataGridColumn dataField="COMMON"/>
        <mx:DataGridColumn dataField="PRICE"/>
        <mx:DataGridColumn dataField="AVAILABILITY"/>
        <mx:DataGridColumn dataField="LIGHT"/>
    </mx:columns>
</mx>DataGrid>

```

19.6 Generate XML Objects from Arrays

Problem

You need to generate an XML object from an array.

Solution

Use the index of the array to access the data within the array and create an XML object from each data object. Then call the `appendChild` method of the XML to add the newly created node to the main XML.

Discussion

Adding nodes to an XML document is done through the `appendChild` method of the XML class. However, to add properties to the XML object, you use the binding tag operators (`{}`) to populate the value of a node or an attribute:

```
var arr:Array = [1, 2, 3, 4, 5];

var xml:XML = new XML(<data></data>);

for(var i:int = 0; i<arr.length; i++) {
    xml.appendChild(<id>{arr[i]}</id>);
}
```

Set the value of the `<id>` tag to the value of the Array at the index and append that value to the XML object:

```
var ids:Array = [121, 122, 123];
var names:Array = [{"Murphy", "Pat"}, {"Thibaut", "Jean"}, {"Smith", "Vijay"}]
var x:XML = new XML("<employeeList></employeeList>");

for (var i:int = 0; i < ids.length; i++)
{
    var newnode:XML = new XML();
    newnode =
        <employee id={ids[i]}>
            <last>{names[i][0]}</last>
            <first>{names[i][1]}</first>
        </employee>;

    x = x.appendChild(newnode)
}
```

19.7 Handle Namespaces in XML Returned by a Service

Problem

You need to parse XML returned from a web service that contains a custom namespace and extension.

Solution

Declare a `namespace` variable, set it to the location of the namespace for the returned XML, and then call the `use` method with that namespace before doing any XML processing.

Discussion

Parsing XML that contains a custom namespace is difficult and requires that the namespace be declared before any XML is returned and parsed from within that namespace. For example, many web services return XML that contains a namespace declaration such as the following:

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:PriceResult>
      <m:Price>34.5</m:Price>
    </m:PriceResult>
  </soap:Body>

</soap:Envelope>
```

The namespace declared in this line

```
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
```

must be declared before the XML is processed. This is done by declaring the namespace as shown here:

```
private namespace w3c = "http://www.w3.org/2001/12/soap-envelope";
use namespace w3c;
To access the Price node of the above SOAP response, use the namespace title in the
E4X statement as shown here:var prices:XMLList = xml.m::PriceResult.m::Price;
```

Using the `.` operator to access the child of any XML response that uses a qualified namespace requires that the namespace is declared and followed by the `::` operator and then the node name. For the following XML object

```
<m:PriceResult>
  <m:Price>34.5</m:Price>
</m:PriceResult>
```

the `price` node would be accessed as follows:

```
m::PriceResult.m::Price
```

19.8 Encode an ActionScript Data Object as XML

Problem

You need to convert an ActionScript object into XML.

Solution

Use the `SimpleXMLEncoder.encodeValue` method to write an object and all of its properties to an `XMLDocument`.

Discussion

The `SimpleXMLEncoder` object is very useful when creating XML to send to a web service or to the URL of a server-side method that expects XML. The `SimpleXMLEncoder` object defines an `encodeValue` method with the following signature:

```
encodeValue(obj:Object, qname:QName, parentNode:XMLNode):XMLNode
```

The generated XML is returned by the method, but is also attached to the `XMLNode` within the `XMLDocument` object to which the `parentNode` is attached. This method requires that the legacy `XMLDocument` object have the generated XML attached to it. After the `XMLDocument` has been generated, though, it can be converted to an XML object by calling the constructor of the XML object and passing the document as an argument to the constructor:

```
var doc:XMLDocument = new XMLDocument('<data></data>');  
var xml:XML = new XML(doc);
```

A complete code listing that will encode an object into an XML document is shown here:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300"  
creationComplete="init()">  
  <mx:Script>  
    <![CDATA[  
      import mx.rpc.xml.SimpleXMLEncoder;  
  
      private var o:Object = {name:"Josh", description_items:{height:'183cm',  
weight:'77k'}}};  
  
      private var doc:XMLDocument;  
  
      private function init():void {  
  
          doc = new XMLDocument('<data></data>');  
          var simpleEncode:SimpleXMLEncoder = new SimpleXMLEncoder(doc);  
          var node:XMLNode = simpleEncode.encodeValue(o, new QName('http://local  
host/ns/ws', 'ls'), doc.firstChild);  
  
      }  
  
    ]]>  
  </mx:Script>  
</mx:Canvas>
```

After the `SimpleXMLEncoder.encodeValue` method is called, the `XMLDocument` object will have the following structure:

```

<data>
  <obj>
    <description_items>
      <height>183cm</height>
      <weight>77k</weight>
    </description_items>
    <name>Josh</name>
  </obj>
</data>

```

19.9 Populate a Component with Complex XML Data

Problem

You need to visually maintain the correct hierarchy when displaying XML that contains multiple nested child nodes.

Solution

Use the `mx.controls.Tree` control or the `AdvancedDataGrid` control to display the data. Create a `HierarchicalData` object and pass the XML to it to ensure that the `Tree` or `AdvancedDataGrid` will be able to properly navigate the XML structure.

Discussion

To display the following XML describing a menu in a `Tree` or an `AdvancedDataGrid`, you can use the `HierarchicalData` object to ensure that the data is displayed properly.

```

<mx:XMLList id="foodXML">
  <menu label="Menu">
    <breakfast_menu label="Breakfast">
      <food label="Eggs and Homefries" price="$6.95" description="Two eggs,
homefries, toast, coffee"/>
      <food label="Item" name="Strawberry Belgian Waffles" price="$7.95"
description="light Belgian waffles with strawberries and whipped cream"/>
    </breakfast_menu>
    <lunch_menu label="Lunch">
      <food label="Soup and Sandwich" price="$8.95" description="Sandwich served
with hot soup"/>
    </lunch_menu>
  </menu>
</mx:XMLList>

```

The `HierarchicalData` object wraps XML or nested data objects to provide the following methods that can be used by a display control to properly display the relationship between nodes in XML:

`canHaveChildren(node:Object):Boolean`

Returns true if the node can contain children.

`getChildren(node:Object):Object`

Returns an object representing the node's children.

`getData(node:Object):Object`

Returns data from a node.

`getRoot():Object`

Returns the root data item.

`hasChildren(node:Object):Boolean`

Returns true if the node has children.

To wrap the XML in a `HierarchicalData` object, pass the XML to the constructor as shown here:

```
var hData:HierarchicalData = new HierarchicalData(myXMLObj);
```

In the following example, the `menuList` XMLList item is passed to a `HierarchicalData` object and displayed within an `AdvancedDataGrid` component:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300"
creationComplete="init()">
<mx:Script>
  <![CDATA[

    import mx.controls.advancedDataGridClasses.AdvancedDataGridColumn;
    import mx.collections.HierarchicalData;

    private function init():void {

    }

    private function labelFunc(item:Object, column:AdvancedDataGridColumn):String
    {
      if(String(item.@label) != "") {
        return String(item.@label);
      }
      return "";
    }

  ]]>
</mx:Script>
  <mx:AdvancedDataGrid dataProvider="{new HierarchicalData(foodXML)}" width="100%"
height="100%">
    <mx:columns>
      <mx:AdvancedDataGridColumn dataField="name" labelFunction="labelFunc"/>
      <mx:AdvancedDataGridColumn dataField="@description"/>
      <mx:AdvancedDataGridColumn dataField="@price"/>
    </mx:columns>
  </mx:AdvancedDataGrid>
</mx:VBox>
```

19.10 Decode XML from a Web Service into Strongly Typed Objects

Problem

You need to transform an XML object or an XMLList object into one or more strongly typed objects.

Solution

Use the `SchemaTypeRegistry.registerClass` method to register the class by using the qualified Namespace and the `SimpleXMLDecoder` class to decode the XML into objects.

Discussion

The `SchemaTypeRegistry.registerClass` method lets you register a class based on a type that will be returned from a web service. The class must be described in the WSDL file, where the web service describes all its methods and types. For example, all properties of an object named `Plant` are defined here:

```
<types>
  <xsd:schema
    targetNamespace=
      "http://localhost/ns/ws"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:complexType name="Plant">
      <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" name="common"
          nillable="true" type="xsd:string"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="botanical"
          nillable="true" type="xsd:string"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="zone"
          nillable="true" type="xsd:string"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="light"
          nillable="true" type="xsd:string"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="price"
          nillable="true" type="xsd:string"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="availability"
          nillable="true" type="xsd:int"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>
</types>
```

The method that will return the `Plant` object is defined like this:

```
<binding name="PlantService" type="tns:Plant">
  <soap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="getPlants">
```

```

<soap:operation soapAction="getPlants"/>
<input>
  <soap:body use="encoded"
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://localhost/ns/ws"/>
  <soap:header message="tns:getPlants" part="header" use="literal"/>
</input>
<output>
  <soap:body use="encoded"
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://localhost/ns/ws"/>
</output>
</operation>

```

Note that the type of the service is defined as `tns:Plant`, indicating that the service will return `Plant` objects as defined in the preceding code. The `SchemaTypeRegistry` uses this declaration to map the web service `Plant` object to an `ActionScript` representation of the data in that object. The method requires a qualified `Namespace` object and a `Class` object that contains the `ActionScript` representation of the class.

```

var qname:QName = new QName("http://localhost/ns/ws", "Plant");
mx.rpc.xml.SchemaTypeRegistry.getInstance().registerClass(qname, Plant);

```

The `Plant` class that will be generated is a simple value object with public properties representing the data required for the `Plant` object:

```

package oreilly.cookbook
{
    public class Plant
    {
        public var common:String;
        public var botanical:String;
        public var zone:String;
        public var light:String;
        public var price:String;
        public var availability:String;
    }
}

```

After the type has been registered, the resulting object of the web service will be cast as type `Plant`, allowing you to work with strongly typed objects without the use of an AMF service.

```

<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300"
creationComplete="init()">
  <mx:WebService id="ws" wsdl="http://localhost/service.php?wsdl" result="trace(event.result)"/>
  <mx:Script>
    <![CDATA[
      import mx.rpc.events.ResultEvent;
      import mx.rpc.xml.SchemaTypeRegistry;
      import mx.rpc.xml.QualifiedResourceManager;
      import mx.rpc.xml.SimpleXMLDecoder;
    ]]>

```

```
private function init():void {  
  
    var qname:QName = new QName("http://localhost/ns/ws", "Plant");  
    mx.rpc.xml.SchemaTypeRegistry.getInstance().registerClass(qname, Plant);  
}  
  
    ]]>  
    </mx:Script>  
</mx:Canvas>
```