

ActionScript 3.0 includes a group of classes based on the ECMAScript for XML (E4X) specification (ECMA-357 edition 2). These classes include powerful and easy-to-use functionality for working with XML data. This article is an introduction to working with XML in AS3.

For more information about the ECMAScript for XML (E4X) read the full specifications of this powerful sub-language at: <http://www.ecma-international.org/publications/standards/Ecma-357.htm>

## Requirements

In order to make the most of this article, you need the following files:

<http://thecodewriter.com/pub/sourcecode/xmlbasics.zip>



This work is licensed under a Creative Commons Attribution 3.0 Unported License.

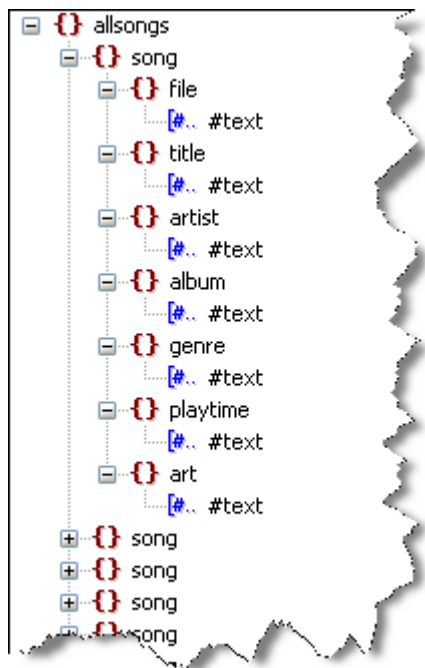
<http://creativecommons.org/licenses/by/3.0/>

## The XML Structure

XML is a standard way of representing structured information so that it is easy for computers to work with and reasonably easy for people to write and understand. XML is an abbreviation for eXtensible Markup Language. The XML standard is available at [www.w3.org/XML/](http://www.w3.org/XML/).

XML uses a tree structure (with branches and leaves known as nodes and values) which offers a standard and convenient way to categorize data, to make it easier to read, access and manipulate.

See the following representation:



The following is a simple example of XML data:

### Sample XML File

```
<?xml version='1.0' encoding='utf-8' ?>
<allsongs>
  <song>
    <file>../mp3s/01 Don't Know Why.mp3</file>
    <title>Don't Know Why</title>
    <artist>Norah Jones</artist>
    <album>Come Away With Me</album>
```

```
<genre>Pop</genre>
<playtime>3:06</playtime>
<art>ComeAwayWithMe.jpg</art>
</song>
<song>
  <file>../mp3s/01 Indescribable.mp3</file>
  <title>Indescribable</title>
  <artist>Chris Tomlin</artist>
  <album>Arriving</album>
  <genre>Inspirational</genre>
  <playtime>3:58</playtime>
  <art>Arriving.jpg</art>
</song>
</allsongs>
```

## AS3 XML Classes

AS3 includes four (4) E4X classes: XML, XMLList, QName and Namespace. However, the two main classes are XML and XMLList.

- **XML:** Represents a single XML element, which can be an XML document with multiple children or a single-value element within a document. *See the `xmlDataFile.xml` listing above.*
- **XMLList:** Represents a set of XML elements. An XMLList object is used when there are multiple XML elements that are "siblings" (at the same level and contained by the same parent in the XML document's hierarchy). Consider the following example:

### XMLList Sample

```
<song>
  <file>../mp3s/01 Don't Know Why.mp3</file>
  <title>Don't Know Why</title>
  <artist>Norah Jones</artist>
  <album>Come Away With Me</album>
  <genre>Pop</genre>
  <playtime>3:06</playtime>
  <art>ComeAwayWithMe.jpg</art>
</song>
```

## Declaring XML as Literals

Often, XML data is loaded from an external source, however assigning XML data as literals within the ActionScript code is also possible.

```
var xmlData:XML = <books>
    <book isbn="059652787X">
        <title>Learning ActionScript 3.0</title>
        <author>Rich Shupe</author>
    </book>
    <book isbn="1590598458">
        <title>Object-Oriented ActionScript 3.0</title>
        <author>Todd Yard</author>
    </book>
</books>;
```

## Loading External XML Files

Loading external XML files is a three step process handled by the URLLoader class. The process can be described as follows:

1. Create a new instance of the URLLoader class.
2. Set a listener to receive the "loader" COMPLETE event.
3. Load the external XML file.

Consider the following code:

```
1. var loader:URLLoader = new URLLoader();
2. loader.addEventListener(Event.COMPLETE, OnLoaded);
3. loader.load(new URLRequest("xmlDataFile.xml"));
```

In the code above, a new instance of the URLLoader class is created (line 1), a listener is set to receive the COMPLETE event which gets fired after the file has finished loading (line 2).

Note that a function to receive the COMPLETE event is required, in this case the **OnLoaded** function (which is defined latter on).

Next, the `URLLoader.load()` method starts loading in the external XML data (line 3). For this a new `URLRequest` object with the path (string) to the XML file is passed to the `URLLoader.load()` method.

*Remember:*

When downloading data from the internet, the data is downloaded piecewise as streams. So, the `URLRequest` class ensures that all of the data is loaded in its entirety before calling the **OnLoaded** function, that way the XML data is passed to the `OnLoaded` function all at once.

Next, the **OnLoaded** function is defined:

```
function OnLoaded(e:Event):void {
    var xmlData:XML = new XML(e.target.data);
}
```

The event object sent to the function contains, among other things, the actual data from the loaded file. This can be retrieved using the `e.target.data` property. Finally, all of the data is stored in the XML object called `xmlData`.

*Note:*

Trying to load or initiate non valid XML (for example trying to load an XML file with an unclosed tag) will throw an exception.

The following code will catch the exception:

```
public function OnLoaded(e:Event):void {
    try
    {
        var xmlData:XML = new XML(e.target.data);
    }
    catch(err:Error)
    {
```

```
        trace("Error: " + e.message);
        return;
    }
}
```

## Manipulating the XML Data

The dot (.) operator and the descendent accessor (..) operator allow accessing child properties (parent and child nodes) of an XML object. The @ symbol (the attribute identifier operator) allows access to attributes.

Keep in mind that when accessing nodes and attributes of an XML object we get an XMLList object in return, so the result must be assigned to an XMLList type variable.

## Code Examples

There are about 21 methods to work with (among other things) the hierarchical structure, attributes and properties of XMLList objects, a handful of them are shown in the following examples (*use the xmlDataFile.xml included in the zip file for reference*):

### First: Setup the class

package

```
{
    import flash.display.MovieClip;
    import flash.events.Event;
    import flash.net.URLLoader;
    import flash.net.URLRequest;

    public class xmlBasics extends MovieClip
    {
        public function xmlBasics()
        {
            var loader:URLLoader = new URLLoader();
            loader.addEventListener(Event.COMPLETE, OnLoaded);
        }
    }
}
```

```
        loader.load(new URLRequest("xmlDataFile.xml"));
    }

    public function OnLoaded(e:Event):void {
        var xmlData:XML = new XML(e.target.data);
    }
}
}
```

## The Most Basic

Let's say you want information related to just one artist.

*Add the following code to the OnLoaded function:*

```
var oneArtist:XMLList = xmlData.song.(artist == "Norah Jones");
trace(oneArtist);
```

*The code above will return:*

```
<song>
  <file>../mp3s/01 Don't Know Why.mp3</file>
  <title>Don't Know Why</title>
  <artist>Norah Jones</artist>
  <album>Come Away With Me</album>
  <genre>Pop</genre>
  <playtime>3:06</playtime>
  <art>ComeAwayWithMe.jpg</art>
</song>
<song>
  <file>../mp3s/03 Those Sweet Words.mp3</file>
  <title>Those Sweet Words</title>
  <artist>Norah Jones</artist>
  <album>Feels Like Home</album>
  <genre>Pop</genre>
  <playtime>3:23</playtime>
  <art>FeelsLikeHome.jpg</art>
```

```
</song>
<song>
  <file>../mp3s/05 Come Away With Me.mp3</file>
  <title>Come Away With Me</title>
  <artist>Norah Jones</artist>
  <album>Come Away With Me</album>
  <genre>Pop</genre>
  <playtime>3:18</playtime>
  <art>ComeAwayWithMe.jpg</art>
</song>
```

Now, if you want only the song titles, do the following:

*Change:*

```
var oneArtist:XMLList = xmlData.song.(artist == "Norah Jones");
```

*To:*

```
var oneArtist:XMLList = xmlData.song.(artist == "Norah Jones").title;
```

*Or:*

```
trace(oneArtist);
```

*To:*

```
trace(oneArtist.title);
```

*In either case the result is:*

```
<title>Don't Know Why</title>
```

```
<title>Those Sweet Words</title>
```

```
<title>Come Away With Me</title>
```

If you want to remove the “<title></title>” tags, use `.text()`, for example:

*Change:*

```
trace(oneArtist.title);
```

*To:*

```
trace(oneArtist.title.text());
```

*...and the result is:*

Don't Know Why  
Those Sweet Words  
Come Away With Me

In case you need to loop thru the values, try the following code (I will add the values to an array just to expand possibilities):

```
var oneArtist:XMLList = xmlData.song.(artist == "Norah Jones");
var songTitles:Array = new Array()

for(var keepCount:uint=0; keepCount<oneArtist.length(); keepCount++)
{
    songTitles.push(oneArtist.title.text()[keepCount]);
    trace(songTitles[keepCount]);
}
```

*The code above returns:*

Don't Know Why  
Those Sweet Words  
Come Away With Me

To filter more than one tag, try the following code:

```
var oneArtistAlbum:XMLList = xmlData.song.(artist == "Norah Jones" && album
== "Come Away With Me");
trace(oneArtistAlbum.title);
```

*...the result is:*

```
<title>Don't Know Why</title>
```

```
<title>Come Away With Me</title>
```

## A Little More Complex

You can pass the node name and the node value to filter as variables. Try the following code:

```
var filteredData:XMLList;

var filterOptions:Array = [
    {theNode:"artist", theValue:"Kevin MacLeod"},
    {theNode:"title", theValue:"porch blues"}
]

for(var keepCount:uint=0; keepCount<filterOptions.length; keepCount++)
{
    var nodeToFilter:String= filterOptions[keepCount].theNode;
    var valueToFilter:String = filterOptions[keepCount].theValue;

    filteredData = xmlData.song.(child(nodeToFilter) == valueToFilter);
}
trace(filteredData.file);
```

*The code above returns:*

```
../mp3s/porch blues.mp3
```

If you want a list of all the artist, you can use the descendent accessor (..) operator or the descendants() method. For example, if you try the following code:

```
var allArtist:XMLList;
```

```
allArtist = xmlData..artist;
```

```
trace(allArtist);
```

*Or:*

```
allArtist = xmlData.descendants().artist;
```

```
trace(allArtist);
```

*In both cases you will get the following result:*

```
<artist>Norah Jones</artist>
```

```
<artist>Chris Tomlin</artist>
```

```
<artist>Ludwig Van Beethoven</artist>
```

```
<artist>Hootie and The Blowfish</artist>
```

```
<artist>Norah Jones</artist>
```

```
<artist>Audio Adrenaline</artist>
```

```
<artist>Norah Jones</artist>
```

```
<artist>Audioslave</artist>
```

```
<artist>Pearl Jam</artist>
```

```
<artist>Bob Marley and The Wailers</artist>
```

```
<artist>Led Zeppelin</artist>
```

```
<artist>Hootie and The Blowfish</artist>
```

```
<artist>The Doors</artist>
<artist>The Doors</artist>
<artist>Pink Floyd</artist>
<artist>Heroes Del Silencio</artist>
<artist>Barlow Girl</artist>
<artist>Bob Marley and The Wailers</artist>
<artist>Pink Floyd</artist>
<artist>The Doors</artist>
<artist>Pink Floyd</artist>
<artist>Kevin MacLeod</artist>
<artist>Kevin MacLeod</artist>
<artist>Kevin MacLeod</artist>
<artist>Kevin MacLeod</artist>
```

As expected, we get a list of all the artists in the XML file (including duplicates).

It is often necessary to generate a **unique** list from an XML document, a unique list of artists as in this case for example. Unfortunately there is no method in AS3 to do that, however, that can be accomplished with just a few lines of code. Consider the following example:

*Add the following code to the OnLoaded function:*

```
var uniqueList:XMLList = new XMLList();
xmlData.song.(uniqueList = GetUniqueValues(artist, uniqueList));
trace(uniqueList);
```

...and create the **GetUniqueValues** function, the code is listed below.

```
public function GetUniqueValues(theValue:Object, theList:XMLList):XMLList {
    if(!theList.contains(theValue))
    {
        theList += theValue;
    }
    return theList;
}
```

The code above returns:

<artist>Norah Jones</artist>

<artist>Chris Tomlin</artist>

<artist>Ludwig Van Beethoven</artist>

<artist>Hootie and The Blowfish</artist>

<artist>Audio Adrenaline</artist>

<artist>Audioslave</artist>

<artist>Pearl Jam</artist>

<artist>Bob Marley and The Wailers</artist>

<artist>Led Zeppelin</artist>

<artist>The Doors</artist>

<artist>Pink Floyd</artist>

<artist>Heroes Del Silencio</artist>

<artist>Barlow Girl</artist>

<artist>Kevin MacLeod</artist>

...a unique list of artist.

## Conclusion

The examples above hopefully illustrate how to approach common XML related tasks, manipulating fairly simple XML data. Webservices or public APIs, however, often contain elements or attributes in namespaces which require a more elaborate approach but that is the topic for another article.

Hopefully the information helped.

## Reference

Some of the information compiled in this article was collected from the following sources:

Flash CS3 Online documentation

[http://livedocs.adobe.com/flash/9.0\\_es/main/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs\\_Parts&file=00000123.html#wp397047](http://livedocs.adobe.com/flash/9.0_es/main/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Parts&file=00000123.html#wp397047)

Josh Tynjala Blog

<http://joshblog.net/2007/05/08/methods-to-filter-data-with-e4x-in-flash-9/>

E4X in ActionScript 3.0

<http://sephiroth.it/tutorials/flashPHP/E4X/index.php>

Using XML in Flash CS3/AS3 - Page 1

[http://kirupa.com/developer/flashcs3/using\\_xml\\_as3\\_pg1.htm](http://kirupa.com/developer/flashcs3/using_xml_as3_pg1.htm)