Loading and Using XML Data to drive Dynamic Content

In this unit, you will learn how to consume XML content in the Flash Lite player.

Adobe

Objectives

After completing this unit, you should be able to:

- Understand XML formatted data
- Understand `Array` and `Object` aggregate variables and syntax
- Load XML into the Flash Lite Player
- Understand the security sandbox model in the Flash Lite player
- Deal with the asynchronous nature of XML loading in Flash Lite
- Parse XML Data using the Flash Lite `XML` class
- Loop over XML data

Using XML

An XML document consists of:

- ▶XML declaration
- ▶Root element
- ▶Child elements
- ▶Text elements
- ▶Element attributes

```
<?xml version="1.0" encoding="UTF-8"?>
<bikes>
 <bike id="1">Kona</bike>
 <bike id="2">VooDoo</bike>
 <bike id="3">Rocky Mountain</bike>
</bikes>
```

Note: Node and Element are interchangeable when referring to XML

Using white space in XML files

- Using white space - tabs, returns, etc. - in your files makes the files easier to read and edit. The previous example above used tabs and returns.
  - The following example does not, and has no extraneous white space.

```
<?xml version="1.0" encoding="UTF-8"?><bikes><bike id="1"
 name="Kona"/><bike id="2" name="VooDoo"/><bike id="3"
 name="Rocky Mountain"/></bikes>
```

- If the XML to be loaded includes white space, you must tell Flash Lite to ignore it, or it the XML will not be parsed correctly.
  - This is done by setting the `ignoreWhite` property of the `XML` class to `true`.

Using tag-based markup in XML files

- If your XML text content includes HTML markup, or any other tag-based markup, the text nodes will contain < and > characters which will prevent Flash Lite from correctly parsing the XML.
- Any text nodes containing tag-based markup characters must be escaped directly within your XML using a CDATA tag as shown here. The escaped text must be contained within tags as follows:

```
<![CDATA[ ... ]]>
```


Adobe

Using tag-based markup in XML files

The following examples show HTML markup escaped within a text node using a CDATA tag:

```
<?xml version="1.0" encoding="UTF-8"?>
<bikes>
 <bike id="1">
 <brand>Kona</brand>
  <description><![CDATA[ A <b>back country dual suspension</b>
  bike with a supreme balance of lightweight performance and out
  of bounds strength. <br>Mid-weight (28-30 lb) with 4 in travel
  up front and back keep you fresh for all day epic adventures.
  ]]>
  </description>
 </bike>
</bikes>
```

Loading XML data into Flash Lite

The Flash Lite player supports three core approaches to loading and send data:

- ▶`LoadVars` class
- ▶`XML` class
- ▶`XMLSocket` class

We focus on using XML because of the widespread adoption of this data format for networked application development. We do not cover the use of the `LoadVars` class though it can be useful for smaller bandwidth applications.

Using the XML class

- The first step is to create an `XML` object in ActionScript using the `new` keyword.
- The XML data may be passed to the constructor, if available, or loaded from an external file once the object has been created.

Loading inline XML

One approach to creating a parseable (readable) `XML` object is to pass the XML data to the class constructor, as a `String`.

```
var bikeData:String = "<?xml version='1.0' encoding='UTF-
  8'?><bikes><bike id='1' name='Kona'/><bike id='2'
  name='VooDoo'/><bike id='3' name='Rocky Mountain'/></
  bikes>";
var xBikes:XML = new XML(bikeData);
```

Note: When writing XML as a literal `String` *value, delimit attribute values with single quotes ("tick marks") and not double-quotes, as those delimit the* `String` *itself.*

Loading external XML

Before loading external XML data, determine whether you need to set the `ignoreWhite` property of your `XML` object to `true`, because the loaded data includes white space for readability.

```
var xBikes:XML = new XML();
xBikes.ignoreWhite = true;
```

Loading external XML

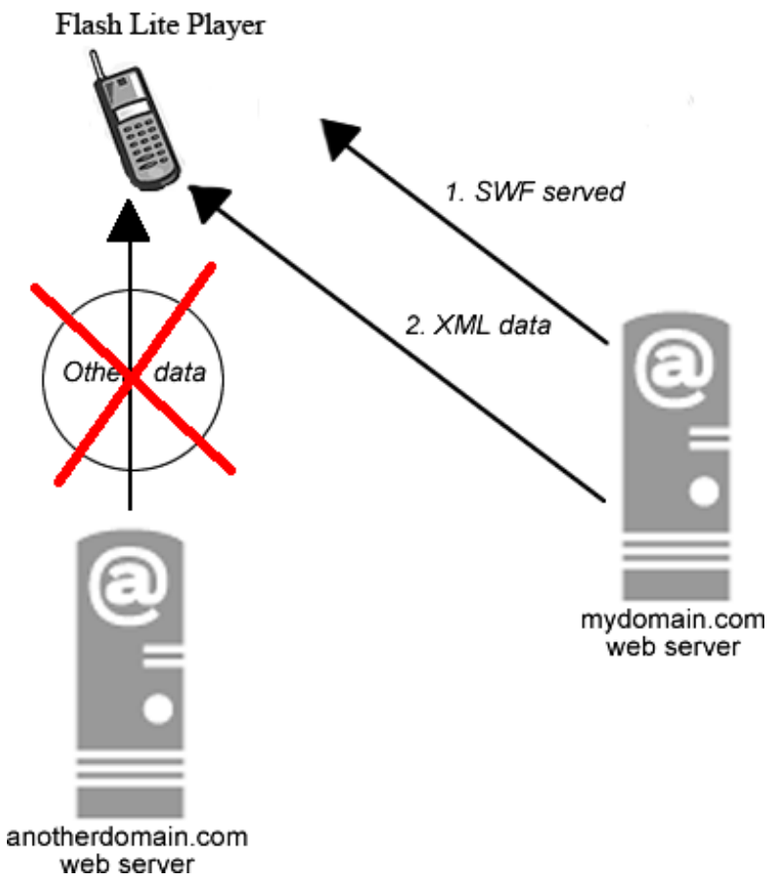
Then load the XML into the `XML` object by passing the path to the XML file as a parameter to the `load()` method, as shown here.

```
// relative path to XML file in same domain
xBikes.load("assets/bikes.xml");

// absolute path to XML file in external domain
// (cross-domain security implicated)
xBikes.load("http://www.adobe.com/bikes.xml");

// absolute path to XML generating script
// (cross-domain security implicated)
xBikes.load("http://www.adobe.com/coldfusion
  /getBikeInfoXml.cfm");
```


Adobe

Making HTTP Requests to Different Domains

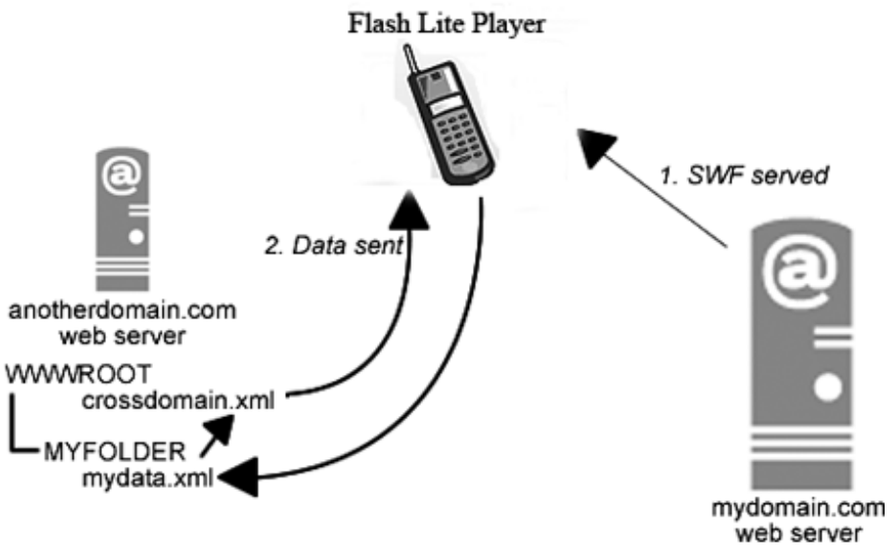
By default, the Flash Lite player does not allow an application to request XML data - or any content of any form - from a domain other than the one from which the requesting SWF was served.



Adobe

Deploying a cross-domain policy file

- If you need to request XML data from a different domain which is almost always in mobile device development:
- You must deploy a cross domain policy file on the server from which the XML data will load. This file needs to be placed at the root of the web server that will serve the HTTP request for the XML file.



The Flash Lite security model

Adobe

Deploying a cross-domain policy file

- A cross domain policy file must be saved with the specific file name `crossdomain.xml` in the web root directory of the server from which the SWF running in a Flash Lite player will request the XML data (or any other asset).
- Cross domain policy files may restrict access to specific domains or IP addresses, and may use wildcards such as an *.

```
<?xml version="1.0"?>
<!-- http://www.foo.com/crossdomain.xml -->
<cross-domain-policy>
 <allow-access-from domain="www.adobe.com" />
 <allow-access-from domain="*.foo.com" />
 <allow-access-from domain="105.216.0.40" />
</cross-domain-policy>
```

Handling the asynchronous response from XML requests

- Flash Lite makes an asynchronous request to the server using the `XML` document object.
  - You must wait for the server to send the results back to the Flash Lite player.
  - You cannot use the loaded XML immediately.
  - When the requested data has been fully loaded and parsed, the `onLoad` event of the `XML` object is broadcast. The `onLoad` event is also fired if the XML file fails to load.

onLoad Code example

```
var bikes:XML = new XML();
bikes.load("assets/bikes.xml");
bikes.onLoad = function(success:Boolean):Void
{
 if(success)
 {
  // do something with XML data
 }
 else
 {
  // give error message
 }
}
```

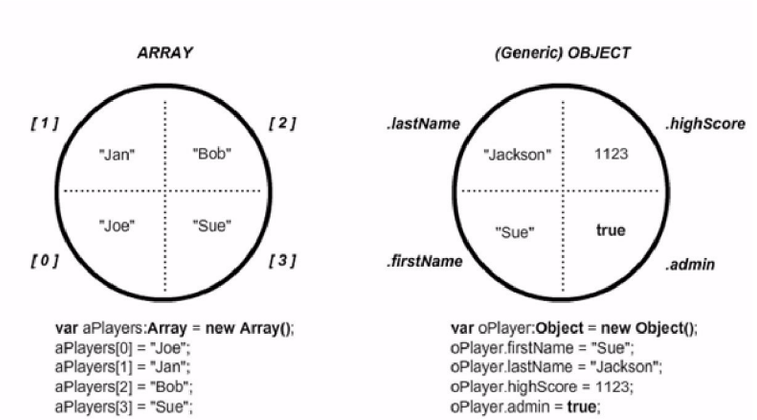
Walkthrough 1: Loading XML

In this walkthrough, you will perform the following tasks:

- Migrate the timeline code to a class file
- Review the contents of the `clues.xml` file
- Create an `XML` object
- Call the `load()` function on the `XML` object
- Create an `onLoad` event handler for the `XML` object

Understanding Array and Objects variables

- An aggregate variable is one which holds several distinct values, rather than just one.
- ActionScript supports two classes of objects which may be used as aggregate variables: `Array` object and `Object` objects.



`Array` *and* `Object` *aggregate variables*

Adobe

Creating and using an Array variable

- An array is useful when multiple pieces of information with an inherent sequence must be used together.For example, dates in a calendar, or songs to be played in order.
- There are three ways to create an array.
  - Instantiating and directly assigning values
  - Instantiating with initial values
  - Using literal Array notation

Instantiating and directly assigning values

- The first value in an array is at index 0.
- Although values can be assigned to any index, is the best practice to store values sequentially within an array, beginning at 0.

```
var colors_array:Array = new Array();
colors_array[0] = 0xFF0000;
colors_array[1] = 0x00FF00;
colors_array[2] = 0x0000FF;
```

Instantiating with initial values

- The `Array` class constructor may be passed a series of initial values, to populate the array.

```
var colors_array:Array = new Array(0xFF0000, 0x00FF00,
  0x0000FF);
```

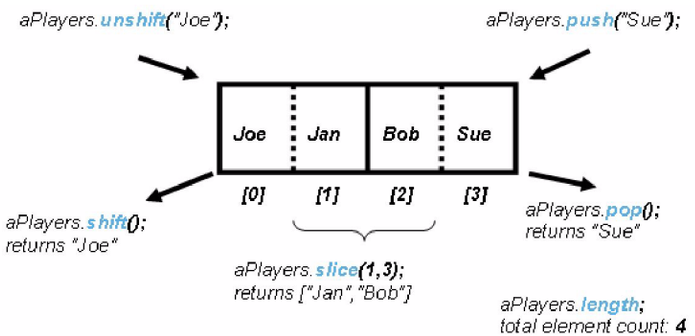
Using literal Array notation

- An array may be created without explicitly creating a `new Array` object, using array literal notation.

```
var colors_array:Array = [0xFF0000,0x00FF00,0x0000FF];
```

Manipulating data within an Array

- The `Array` class provides a property, and several methods, for counting, adding, removing, and extracting subsets from the array.



Ways to manipulate `Array` *data*

- `push(value)`: adds a value to top of the array
- `pop()`: removes and returns top-most value in the array
- `unshift(value)`: adds value to bottom of the array, shifting the rest
- `shift()`: removes and returns the bottom-most value in the array
- `length`: a property containing the count of items in the array

Creating and using a "generic" Object

- Instances of the root class of the ActionScript 2.0 class hierarchy, Object, can have new properties added to them at runtime.
- So, instances of the Object class (sometimes called "generic objects" or "data objects") may be used as aggregate variables holding name/value pairs of data.
- There are two basic ways to create a data object.
  - Instantiating and directly assigning properties
  - Using literal Object notation

Instantiating and directly assigning properties

- An instance of the `Object` class is created, then properties are added to it using either dot or bracket notation, and lastly values are assigned.

```
var book:Object = new Object();

book.title = "The Golden Bough";
book.price = 21.00;
book.inStock = true;

- or -

book["title"] = "The Golden Bough";
book["price"] = 21.00;
book["inStock"] = true;
```

Note: You could also declare a `Book` *class, with relevant properties, for use as a Value Object within your ActionScript code. This approach is not discussed in this course.*

Using literal Object notation

- A data object can be created without explicitly creating a `new Object`, using Object literal notation.

```
var book:Object = {title:"The Golden Bough", price:21.00,
  inStock:true};
```

Understanding Array and Object syntax

It is important to understand the syntax used in referring to `Object` properties - dot or bracket notation - and to `Array` elements - bracket notation - as this syntax is used inherently by other ActionScript classes, particularly the `XML` class.

Parsing and using XML data

- The Flash Lite player contains no automatic way to parse and map XML files into native ActionScript data structures.
- You must load the XML into an `XML` object.
- The resulting data structure is a nested tree of Objects and Arrays, and both Object and Array syntax are used in accessing the loaded values.

Loading the XML data

When the data has loaded, and is ready to use, the `XML` object broadcasts an `onLoad` event.

```
------- Bikes.xml --------------------
<bikes>
  <bike id="1" price="750">Kona</bike>
  <bike id="2" price="900">VooDoo</bike>
  <bike id="3" price="1100">Rocky</bike>
</bikes>
------- Bikes.xml --------------------

var xml:XML = new XML();
xml.ignoreWhite = true;
xml.load("Bikes.xml");
xml.onLoad = function(success:Boolean):Void{
 if (success == true){
//extract and use the data
}
```

Understanding the XMLNode objects

Each node within the loaded `XML` object is represented by an `XMLNode` object. So, you traverse the data using properties of the `XMLnode` class:

Property	Description
`childNodes`	`Array` of child `XMLNode` objects for the specified node
hasChildNodes	`Boolean` indicating whether the `childNodes` property is populated for the current node
length	count of items in the `childNodes` `Array` for the specific node
`attributes`	Generic object ("associative array") of attributes for the specified node

Property	Description
`firstChild`	Equivalent reference to `childNodes[0]`
`nodeName`	Name of current node
`nodeValue`	Text of specified node if it is a text node
nodeType	Integer reference of the type of the specific node (1 = Element Node, 2 = Attribute Node, 3 = Text Node, ... see documentation for full list)
nextSibling	reference to next `XMLnode` object within the same `Array` of child nodes, `null` if there is no next sibling
previousSibling	reference to previous `XMLnode` object within the same child node `Array`, `null` if no previous sibling

Extracting data from the XML object

- The loaded `XML` object contains a series of nested arrays named for the XML nodes, which contain `XMLNode` objects as the array elements.

Note: In the following code samples, assume the XML expression is being used within an `onLoad` *event handler, as described above. Data must be fully loaded before it is used.*

- The first array within the XML object represents the top-level node in the XML object.

Looping over XML data

The `length` property returns the number of nodes in the referenced array.

```
var nodes:Array = xml.firstChild.childNodes;
var id:Number, price:Number, bikeName:String;
for (var i:Number = 0; i < nodes.length; i++)
{
 id = nodes[i].attributes.id;
 price = nodes[i].attributes.price;
 bikeName = nodes[i].firstChild.nodeValue;
 trace(id + ": " + bikeName + " $" + price);
}
// displays
// 1: Kona $750
// 2: VooDoo $900
// 3: Rocky $1100
```

Walkthrough 2: Parsing XML

In this walkthrough, you will perform the following tasks:

- Create an `Array` to hold the clue data
- Parse the loaded XML and store it in the `Array`
- Apply the data to the view
- Set a variable on the main timeline to track progress

Summary

- XML data is formatted as plain text within properly nested tags.
- XML data may be loaded into the Flash Lite Player via HTTP using the `load()` method of the `XML` class.
- If XML data contains white space for easy readability, the `ignoreWhite` property of the `XML` object must be set to true.
- When XML data has been loaded and parsed, the `XML` object broadcasts an `onLoad` event.
- The `onLoad` event handler is passed a `Boolean` (true or `false`) argument to indicate whether the data successfully loaded .
- Once loaded, the `XML` object holds a nested array of `XMLNode` objects, with the first array representing the root node of the XML document.
- Wherever nodes are repeated as siblings within an XML document, an array will be created in the `XML` object.

Summary

- The `firstChild` and `childNodes[0]` properties of an `XMLNode` object refer to the same object.
- Attributes of an XML node are accessed through the data object referred to by the `attributes` property of an `XMLNode` object.
- Text content within an `XMLNode` is treated as its own distinct `XMLNode` object.
- The content of a text node can be accessed through that nodes' `nodeValue` property.
- Repeated nodes within an XML object can be iterated ("looped over") using `for`, `while`, or `do .. while` loops.