CS 176
Advanced Scripting

Working With XML

Elie Abi Chahine

Introduction

What's an XML

XML stands for eXtensible Markup Language.

It 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.

In other words, it offers a standard and convenient way to categorize data, to make it easier to read, access, and manipulate.

Simple XML Example

```
<song>
    <title> What you know? </title>
    <artist> Steve and the flubberblubs </artist>
    <year> 1989 </year>
    <lastplayed> 2006-10-17-08:31 </lastplayed>
</song>
```

More Complex XML Example

```
<album>
    <title>Questions, unanswered</title>
    <artist>Steve and the flubberblubs</artist>
    <year>1989</year>
    <tracks>
        <song tracknumber="1" length="4:05">
            <title>What do you know?</title>
            <artist>Steve and the flubberblubs</artist>
            <lastplayed>2006-10-17-08:31</lastplayed>
        </song>
        <song tracknumber="2" length="3:45">
            <title>Who do you know?</title>
            <artist>Steve and the flubberblubs</artist>
            <lastplayed>2006-10-17-08:35</lastplayed>
        </song>
    </tracks>
</album>
```

Elements

In XML format, data is organized into elements (which can be single data items or containers for other elements) using a hierarchical structure.

Every XML document has a single element as the top level or main item; inside this root element there may be a single piece of information, although there are more likely to be other elements, which in turn contain other elements, and so forth.

Elements are also known as ***nodes.***

Tags

```
<song tracknumber="1" length="4:05">
        <title>What do you know?</title>
        <artist>Steve and the flubberblubs</artist>
        <mood>Happy</mood>
        <lastplayed>2006-10-17-08:31</lastplayed>
</song>
```

• Each element is distinguished by a set of tags—the element’s name wrapped in angle brackets.

•The opening tag, indicating the start of the element, has the element name: ***<title>***

•The closing tag, which marks the end of the element, has a forward slash before the element’s name: ***</title>***

•If an element contains no content, it can be written as an empty element (sometimes called a self-closing element): ***<lastplayed/>*** which is identical to ***<lastplayed> </lastplayed>***

Attributes

```
<song tracknumber="1" length="4:05">
        <title>What do you know?</title>
        <artist>Steve and the flubberblubs</artist>
        <mood>Happy</mood>
        <lastplayed>2006-10-17-08:31</lastplayed>
</song>
```

• In addition to the element’s content contained between the opening and closing **tags**, an element can also include other values, known as **attributes**, defined in the element’s opening tag.

• For example, the ***song element*** contains two attributes:

- ➢ ***tracknumber*** with the value ***1***
- ➢ ***length*** with the value ***4:19***

Common XML Tasks

When you work with XML in ActionScript, you are likely to do the following tasks:

- Constructing XML documents (adding elements and values)
- Accessing XML elements, values, and attributes
- Filtering (searching in) XML elements
- Looping over a set of XML elements
- Converting data between XML classes and the String class
- Working with XML namespaces
- Loading external

AS3 & XML

ActionScript XML Classes

ActionScript 3.0 includes several classes that are used for working with XML-structured information. The two main classes are as follows:

•**XML:** Represents a single XML element, which can be an XML document with multiple children or a single-value element within a document.

• **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).
Example:

```
<artist type="composer"> Fred Wilson </artist>
<artist type="conductor"> James Schmidt </artist>
<artist type="soloist"> Susan Harriet Thurndon </artist>
```

In addition to the built-in classes for working with XML, ActionScript 3.0 also includes several operators that provide specific functionality for accessing and manipulating XML data. We will be showing many examples in future slides.

Creating an XML using AS3

```
var xml:XML =
<level>
   <Object name="Hero">
       <Class> GamePlay.Level.HeroSP </Class>
       <Position>
          <X> 100 </X>
          <Y> 50 </Y>
       </Position>
       <Scale>
           <X> 1.5 </X>
           <Y> 1.5 </Y>
       </Scale>
    </Object>
</level>;

trace(xml);
```

Loading an XML using AS3

```
import flash.events.Event;
import flash.net.URLLoader;
import flash.net.URLRequest;

var xml:XML;
var loader:URLLoader = new URLLoader();
var request:URLRequest = new URLRequest("level.xml");
loader.load(request);
loader.addEventListener(Event.COMPLETE, onComplete);

function onComplete(event:Event):void
{
    var loader:URLLoader = event.target as URLLoader;
    if (loader != null)
    {
        xml = new XML(loader.data);
        trace(xml);
    }
    else
    {
        trace("loader is not a URLLoader!");
    }
}
```

Loading an XML using AS3

URLLoader: The URLLoader class downloads data from a URL as text, binary data, or URL-encoded variables. It is useful for downloading text files, XML, or other information to be used in a dynamic, data-driven application.

URLRequest: URLRequest objects are passed to the load() methods of the Loader, URLStream, and URLLoader classes, and to other loading operations, to initiate URL downloads.

Synchronous vs Asynchronous

Executing something synchronously means you wait for it to finish before moving on to another task.

Executing something asynchronously means you can move on to another task before it finishes.

Loading XMLs in AS3 is done asynchronously, that is why we have to listen for the COMPLETE event before using the xml.

Accessing the XML's data

Assume we have the following level.xml file that we loaded into an **xml** instance:

```
<Level>
  <Columns>15</Columns>
  <Rows>11</Rows>
  <TileWidth>50</TileWidth>
  <TileHeight>50</TileHeight>
  <TileIDs>
    <tile id="0" name="Air" class_name="Air" class_type="displayobject"/>
    <tile id="1" name="Ground1" class_name="Ground1" class_type="displayobject"/>
    <tile id="2" name="Ground2" class_name="Ground2" class_type="displayobject"/>
  </TileIDs>
  <TileMap>
    <Row>1,1,1,1,1,1,1,1,1,1,1,1,1,1,1</Row>
    <Row>1,0,0,0,0,0,0,0,0,0,0,0,0,0,1</Row>
    <Row>1,0,0,0,0,0,0,0,1,1,1,1,1,1,1</Row>
    <Row>1,0,0,1,1,0,0,0,0,0,0,0,0,0,1</Row>
    <Row>1,0,0,0,1,0,0,0,0,0,0,0,0,0,1</Row>
    <Row>1,0,0,0,1,0,0,0,0,0,0,0,0,0,1</Row>
    <Row>1,0,0,0,1,0,0,0,0,0,0,0,0,0,1</Row>
    <Row>1,0,0,0,1,0,0,0,0,0,0,0,0,1,1</Row>
    <Row>1,0,0,0,1,1,0,0,0,0,0,0,1,1,1</Row>
    <Row>1,0,0,0,0,0,0,0,0,0,0,1,1,1,1</Row>
    <Row>1,1,1,1,1,1,1,1,1,1,1,1,1,1,1</Row>
  </TileMap>
</Level>
```

Accessing the XML's data

```
trace("**************************************************");
trace(xml.Columns);
trace("**************************************************");
trace(xml.TileIDs);
trace("**************************************************");
trace(xml.TileIDs.tile[0].@name);
trace("**************************************************");
```

Output:

```
**************************************************
15
**************************************************
<TileIDs>
  <tile id="0" name="Air" class_name="Air" class_type="displayobject"/>
  <tile id="1" name="Ground1" class_name="Ground1" class_type="displayobject"/>
  <tile id="2" name="Ground2" class_name="Ground2" class_type="displayobject"/>
</TileIDs>
**************************************************
Air
**************************************************
```

Accessing the XML’s data

```
trace("****************************************************");
trace(xml.TileMap.Row[0]);
trace("****************************************************");
trace(xml.TileMap.Row.length());
trace("****************************************************");
```

Output:

```
****************************************************
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
****************************************************
11
****************************************************
```

Accessing the XML's data

```
var iNumberOfRows:int = xml.TileMap.Row.length();

for(var i:int = 0; i < iNumberOfRows; ++i)
{
    trace(xml.TileMap.Row[i]);
}
```

Output:

```
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,1,1,1,1,1,1,1
1,0,0,1,1,0,0,0,0,0,0,0,0,0,1
1,0,0,0,1,0,0,0,0,0,0,0,0,0,1
1,0,0,0,1,0,0,0,0,0,0,0,0,0,1
1,0,0,0,1,0,0,0,0,0,0,0,0,0,1
1,0,0,0,1,0,0,0,0,0,0,0,0,1,1
1,0,0,0,1,1,0,0,0,0,0,0,1,1,1
1,0,0,0,0,0,0,0,0,0,0,1,1,1,1
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
```

Learning more about XML

The below link contains more information about XML (you don't need them for CS176 but it doesn't hurt to know where to find them in the future):

http://help.adobe.com/en_US/ActionScript/3.0_ProgrammingAS3/

All the information are found in the ***"Working with XML"*** section.

The End ☺