CGT 353: Principles of Interactive and Dynamic Media
Data Integration

Introduction:

- One of the most useful (and essential) features of Flash is the ability to pass data to and from applications.
- Upcoming lectures will deal will integration with LoadVars, XML, and Shared Objects

The Basics:

- Like HTML forms, Flash's input and dynamic text fields are the primary method for user-defined data creation
- As such, they are the primary means of passing data into and out of Flash
- Similarities between Flash and HTML methods are very similar

Understanding Data Sources and Data Formats:

- A **data source** is any place from which Flash can load external data.
- **Data transfer** is the act of retrieving data from or sending data to a source.
- Any data that one plans to load into Flash must be formatted in a specific way.

Other Terminology:

- **External data:** Data that is stored in some form outside of the SWF file, and loaded into the SWF file when needed.
- **URL-encoded variables:** The URL-encoded format provides a way to represent several variables (pairs of variable names and values) in a single string of text. Individual variables are written in the format `name=value`. Each variable (that is, each name-value pair) is separated by ampersand characters, like this: `variable1=value1&variable2=value2`. In this way, an indefinite number of variables can be sent as a single message.
- **Request method:** When a program such as Flash Player or a web browser sends a message (called an HTTP request) to a web server, any data being sent can be

embedded in the request in one of two ways; these are the two *request methods* GET and POST.

- On the server end, the program receiving the request will need to look in the appropriate portion of the request to find the data, so the request method used to send data from ActionScript should match the request method used to read that data on the server.

URL String:

- Data is defined as a string of text

Firstname=Ron&lastname=Glotzbach&website=http://www.tech.purdue.edu/cgt

- This demonstrates the use of the MIME format which is used with loadVariables(), loadVariablesNum(), or the LoadVars class

- An equals sign(=) associates the variable with the value and an ampersand(&) marks the junction between variables

- While this formats supports an unlimited number of variables, only simple variables can be used

- Use escape() and unescape() functions when special characters are in the strings themselves

```
//create the string
var mYString:String = escape ("title = War & Peace");

// send the encoded string to the output panel

trace(myString);
trace(unescape(myString));

//output: title%20%3D%20War%20%26%20Peace
//output: title = War & Peace
```

AS 2.0 – LoadVars and MovieClip Methods:

- **Movieclip methods** - `getURL(), loadVariables(), loadVariablesNum(), loadMovie(), and loadMovieNum().`

- **LoadVars methods** - `load(), send(), and sendAndLoad()`

- **XML methods** - XML are `XML.send()`, `XML.load()`, and `XML.sendAndLoad()`.

- **getURL()** - returns any information to a browser window, not to Flash Player.
- **loadVariables()** - loads variables into a specified timeline or level in Flash Player.
- **loadMovie()** method loads a SWF file into a specified level or movie clip in Flash Player.
- **loadMovieNum()**

When you use loadVariables(), getURL(), or loadMovie(), you can specify several parameters:

- *URL* is the file in which the remote variables reside.

- *Location* is the level or target in the SWF file that receives the variables. (The getURL() function does not take this parameter.)
  .
- *Variables* sets the HTTP method, either GET (appends the variables to the end of the URL) or POST (sends the variables in a separate HTTP header), by which the variables are sent. When this parameter is omitted, Flash Player defaults to GET, but no variables are sent.

AS 2.0 Methods and Actions for Sending Data into and Out of Flash:

- getUrl()
- LoadVariable()
- LoadMovie()
- LoadVars.load()
- LoadVar.send()
- LoadVars.sendAndLoad()
- XML.load()
- XML.send()
- XML.sendAndLoad()
- XMLSocket.send()

XML:

- AS script can be written to extract information from XML documents, allowing XML documents to serve as a database of sorts

- Will discuss this more in upcoming lectures…

Shared Objects:

- The equivalent of Flash cookies and allows you to store object's locally on the users hard drive.
- By using shared objects, you can not only store variables but ANY kind of data object (arrays, XML objects, etc…)
- Especially useful in gaming when you want to store user information such as names, scores, etc…

Sources that Flash can Load Data From:

Text Files:

- Flash can load test (.txt) files containing data formatted using the URL string format.
- Can be loaded using loadVariables() or the load() method of the LoadVars class

Server Side Scripts:

- Placed on ASP, CFML, CGI, or JSP pages and executed by a server…
- Can return data in a number of formats including XML and URL string formats…

XML Files:

- Usually a text file with XML formatted data…
- More on this later…

XML Socket:

- Applications that run on a server and connect several simultaneous users to one another…
- Flash can send or receive information via the socket using the XML format…

GET vs POST:

- Two ways to transfer data from Flash to server-side scripts: GET or POST
- Also used with HTML forms…
- Using GET you're simply concatenating variable name/ values into the URL

  http://www.purdue.edu/login.asp?firstname=Ron&lastname=Glotzbach
- This method is not very secure and also has a 1024-character limitation
- Using POST, the data is contained in the header of the HTTP request, so you can't see it transferred…
- POST also doesn't have a character limit…

Using the LoadVars Class:

- Use the LoadVars class when working with data in the URL string format…
- Enables you to load variable from a text file or to and from a server-side script…
- Note: You cannot directly write to a text file from Flash…

Creating a LoadVars Object:

var RonInfo:LoadVars = newLoadVars();

Loading variables from a URL into a LoadVars Object:

RonInfo.load(http://www.purdue.edu/stupidtext.text);

So, if you had a LoadVars object named RonInfo and loaded the following string from a text file:

Firstname=Ron&lastname=Glotzbach&age=21

You could reference those names using:

RonInfo.firstname
RonInfo.lastname
RonInfo.age

or

var userAge: Number = RonInfo.age

- If you want to send variables in a LoadVars object to a server side script for processing, use the send() method.

    myLoadVarsObject.send("http://www.purdue.edu/process.asp);

- Note that using **.send** does not get a response back

- Use the **sendAndLoad** method to get a response

myLoadVarsObject.sendAndLoad("http://www.purdue.edu/process.asp, receivingLoadVarsObject);

- If you want the same object that is doing the sending to receive variables, just use the load() method

- Can also use the toString() method of the LoadVars class to create URL-formatted strings

- So, using the previous example:

RonInfo.toString() would give you firstname=Ron&lastname=Glotzbach&age=21

Properties of the LoadVars Class:

1. **contentType –** gives you the mime type specified in the HTTP header
2. **loaded –** returns a true or false value depending on whether or not the data has finished loading into the object

- The only event available to the LoadVars class is the **onLoad** method
- Used to call a function when data has finished loading into an object
- To load variables into an object then call a function when the loading is complete:

    1. Define a function
    2. Create a new LoadVars object
    3. Specify the function to be called when loading complete
    4. Invoke the load()method of the LoadVars object

```
function myFunction():Void{
        trace("Data is loaded");
}
```

```
var container:LoadVars = new LoadVars(); //creates the LoadVars object
container.onLoad = myFunction;  //calls the function(myFunction) when data is loaded
container.load("http://www.purdue.edu/myfile.asp:);
```

Shared Objects:

- Being able to save data to client machine is essential…
- With traditional Web applications, this is done via **cookies…**
- With Flash, we use **shared objects…**
- User the **SharedObject** class…
- **Local shared objects** allow you to store and retrieve data on client computer…
- **Remote shared objects** can be used with **Flash Communcication Server** (outside the scope of this class)

Creating Shared Objects:

- Process for creating a new shared object and opening and existing shared object is the same…
- Static method getLocal() opens a local shared object of the specified name if it exists.

var lsoPreferences:SharedObject = SharedObject.getLocal("userpreferences");

- File extension for shared object files is **.lso,** but don't include that in the parameter
- Flash either finds the existing file or creates a new one…

Setting Values in Shared Objects:

```
var lsoPreferences:SharedObject = SharedObject.getLocal("userpreferences");
lsopreferences.data.backgroundColor = "red";
lsoPreferences.data.name = "A Reader";
```

- The **data** property is an object in itself which you can assign additional properties
- Properties assigned to the SharedObject will not be saved, only the properties of the data object.

Saving the Shared Object to the Client

- Flash automatically tries to save the shared object when the SharedObject instance is deleted…
    - When player closes
    - When movie is closed or replaced in the player
    - When object is deleted with the delete statement
    - When the object is garbage-collected after it falls out of scope
- Don't rely on automatic saves however, use the flush (method)
- Not only does flush() ensure you save the data, but also allows the user to determine how much data the shared object can store (default is 100K)

Retrieving the Shared Object Data:

- Still use getLocal() method
    - As you will read about and learn in class, there are three basic ways to tie Flash into a robust data source:
        - Flash remoting
        - Web services
        - Direct server-side integration
    - **Flash Remoting** is a proprietary technology and rather expensive…but easily the fastest of the three choices
    - **See Article:** *Choosing Between XML, Web Services, and Remoting for Rich Internet Applications*

AS 3.0 Shared Object Example:

```
var so:SharedObject = SharedObject.getLocal("userHighScore");
 so.data.highScore = new Number(1234567890);
 so.flush();
```

Direct Server-Side Integration:

- The flash.net package contains classes to send and receive data
- As we have already seen, you can load external files with URLLoader and URLRequest

- Then use a specific class to access the data, depending on the type of data that was loaded.

- For example, if the remote content is formatted as name-value pairs, you use the URLVariables class to parse the server results.

- If the file loaded is a remote XML document, you can parse the XML document using the XML class's constructor, the XMLDocument class's constructor, or the `XMLDocument.parseXML()` method

- The advantage here stems from the fact that the code for loading external files is the same whether you use the URLVariables, XML, or some other class to parse and work with the remote data.

- flash.net package also contains classes for other types of remote communication.

    - **FileReference** class for uploading and downloading files from a server

    - **Socket and XMLSocket classes** for communicating directly with remote computers over socket connections

    - **NetConnection and NetStream classes**, used for communicating with Flash-specific server resources (such as Flash Media Server and Flash Remoting servers) as well as for loading video files.

AS 3.0 - Using the URLLoader and URLVariables classes:

- **URLLoader class** uses the AS 3.0 event-handling model, which allows you to listen for such events as `complete, httpStatus, ioError, open, progress, and securityError`

Loading data from external documents:

```
var request:URLRequest = new URLRequest("params.txt");
var loader:URLLoader = new URLLoader();
loader.load(request);
```

or

```
var loader:URLLoader = new URLLoader(new URLRequest("params.txt"));
```

XML Example:

```
package
{
    import flash.display.Sprite;
    import flash.errors.*;
    import flash.events.*;
    import flash.net.URLLoader;
    import flash.net.URLRequest;

    public class ExternalDocs extends Sprite
    {
        public function ExternalDocs()
        {
            var request:URLRequest = new
URLRequest("http://www.[yourdomain].com/data.xml");
            var loader:URLLoader = new URLLoader();
            loader.addEventListener(Event.COMPLETE, completeHandler);
            try
            {
                loader.load(request);
            }
            catch (error:ArgumentError)
            {
                trace("An ArgumentError has occurred.");
            }
            catch (error:SecurityError)
            {
                trace("A SecurityError has occurred.");
            }
        }
        private function completeHandler(event:Event):void
        {
            var dataXML:XML = XML(event.target.data);
            trace(dataXML.toXMLString());
        }
    }
}
```

Communicating with external scripts:

```
var variables:URLVariables = new URLVariables("name=Franklin");
var request:URLRequest = new URLRequest();
request.url = "http://www.[yourdomain].com/greeting.cfm";
request.method = URLRequestMethod.POST;
request.data = variables;
var loader:URLLoader = new URLLoader();
loader.dataFormat = URLLoaderDataFormat.VARIABLES;
loader.addEventListener(Event.COMPLETE, completeHandler);
try
{
    loader.load(request);
}
catch (error:Error)
{
```

```
    trace("Unable to load URL");
}

function completeHandler(event:Event):void
{
    trace(event.target.data.welcomeMessage);
}
```

Sources:

1. ActionScript 3.0 Cookbook by Lott, Schall, and Peters
2. Adobe LiveDocs
3. PHP 5 for Flash by David Powers

XML (extensible markup language) is:

- "a markup language for documents containing structured information." (O'Reilly xml.com)
- self-describing
- well-formed

What is XML?

- Based on SGML (Standard Generalized Markup Language)
- Developed because HTML could only function with predefined tags
- Many languages today are based on XML:
  - XHTML
  - SVG – Sclable Vector Graphics
  - RDF Site Summary (RSS)
  - Wireless Markup Language (WML)
- In it's rawest form, XML is simply data holder
- Used for two functions: **exchanging and storing data**

Components of an XML document

- **declaration:** tells the XML parser in the browser the document is XML
  example: <?xml version="1.0"?>
- **body:** contains elements
- **element:** composed of a start-tag, contents, and an end-tag; the element name is self-describing of its contents

example: <name>Marcellus</name>
format: *<startTag>contents</endTag>*
note: it is important to use "/" in the end-tag for the parser to recognize it as such
note: in Flash elements are referred to as "nodes"

- **element relationships:** elements can contain elements which can contain other elements and so on
  note: see in the XML example how the <title> element is contained within the <object> element

- **parent element:** an element containing an element
  note: in the example the <object> element is parent to the <title> element

- **child element:** an element contained within an element
  note: in the example the <title> element is a child of the <object> element

- **root element:** an XML document can contain only one root element; all elements and their children are contained within the root element

- **element attribute**: additional information placed within an element tag
  example: <object type="image">
  format: <element attributeName="attributeValue">
  note: the attributes value must be placed within quotes to be parsed
  note: attributes are not always necessary, in this example the "type" attribute could be a child element of the <object> element (<type>image</type>)
  note: whether or not to use attributes is entirely up to the developer in how they choose to design and organize their data

XML Example:

The proceeding examples will refer to the following XML code:

```
<?xml version="1.0"?>

<portfolio>
	<design>
		<object type="image">
			<name>AT-ST</name>
			<src>images/atstThumb.jpg</src>
			<description>Rendering of Star Wars AT-ST</description>
		</object>
	</design>
</portfolio>
```

Properties and Methods of Flash XML object

Property	Description
XML.firstChild	Read-only; references the first child in the list for the specified node.
XML.lastChild	References the last child in the list for the specified node.
XML.nextSibling	Read-only; references the next sibling in the parent node's child list.
XML.nodeName	The node name of an XML object.
XML.nodeValue	The text of the specified node if the node is a text node.
XML.previousSibling	Read-only; references the previous sibling in the parent node's child list.

Method	Description
XML.load()	Loads a document (specified by the XML object) from a URL.
XML.attributes *	Returns an associative array containing all of the attributes of the specified node.
XML.childNodes *	Read-only; returns an array containing references to the child nodes of the specified node.

Instantiating an XML object:

```
portfolio_xml = new XML();
        //instantiates a new XML object
portfolio_xml.load("portfolio.xml");
        //loads xml document, in this case portfolio.xml located in the root
        //directory
portfolio_xml.ignoreWhite = true;
        //ignoreWhite property initiliazed as true, ignores empty elements in XML
        //document
portfolio_xml.onLoad = portfolioLoad;
        //calls portfolioLoad function when the portfolio_xml object is loaded
```

Loading XML:

```
var myXML:XML = new XML();
myXML.load ("http://www.purdue.edu/info.xml");
```

To Determine When Loading is Finished:

```
function init(){
```

write function to handle and interpret the XML
}

var myXML:XML = new XML();
myXML.onLoad = init;
myXML.load ("http://www.purdue.edu/info.xml");

Sending XML:

var myXML:XML = new XML(<Message><Text> Starting XML Data</Text></Message>);

myXML.send(http://www.purdue.edu/xmldealer.asp)

All in One:

var URL:String = http://www.purdue.edu/startup.asp;

function init(){
trace(objtoreceive)
}

var xmlToSend:String =
"<Login><Username>rjglotzbach</Username><Password>smartalleck</Password></Login>";
var objToSend:XML = new XML (xmlToSend);
var objToReceive:XML = new XML();
objToReceive.onLoad = init;
objToSend.sendAndLoad(URL, objToReceive);

LoadVars:

- Accomplishes pretty much the same stuff, but with name-value pairs instead of XML

Loading Data:

var lvData:LoadVars = new LoadVars();
lvData.load ("bookResult.txt");
lvData.onLoad = function (bSuccess:Boolean):Void{
if(bSuccess){
trace(this.title);
trace(this.author);

```
}

};
```

Sending Data:

```
var lvData:LoadVars = new LoadVars();
lvData.favoriteColor = “red”;
lvData.favoriteSong = “Frayed Ends of Sanity”;
lvData.send (http://www.myserver.com/cgi-bin/surveyResults.cgi);
```