Week 3

Using XML

Receiving the data in plain text is an obvious limitation. Most servers will respond to your request with data in either XML or JSON format. This week we will see how to use data in either one of these formats. First, XML:

Google provides a function that takes a plain text string and converts it to an XML tree. This function, called GXml.parse() takes the text string as parameter and returns the root node of an XML tree that contains all the the data in the string. We can then search through and traverse the XML tree to find the data we are looking for.

Since an html document is really just a special case of an xml document, many of the functions we use to search through the html document we can also use to search through an xml document. Typically, we can call getElementsByTagName() and then loop through every element in the returned array to find what we are looking for.

For example, if allLoaded() is the callback function that we sent to GDownloadUrl(), the following code searches through the returned xml file to find one element.

```
// Called when the request object is finished loading
function allLoaded(content)
{     var xmlDoc = GXml.parse(content);

      // nodeArray contains all the nodes that have tag "State"
      var nodeArray = xmlDoc.getElementsByTagName("state");

      // nodeArray has all nodes in the 1st State that have tag "City"
      nodeArray = nodeArray[0].getElementsByTagName("city");

      // note that to get the text between the tags you must use
      // firstChild.data
      display = nodeArray[0].firstChild.data;
      toConsole(display);
}
```

The following sections contain a program that downloads an xml file and searches for data inside it.

parseXML.html

Instructions: These files are needed to see how a program parses an xml file: parseXML.html, parseXML.js, cities.xml

parseXML.html

parseXML.js

Instructions: These files are needed to see how a program parses an xml file: parseXML.html, parseXML.js, cities.xml

parseXML.js

cities.xml

Instructions: These files are needed to see how a program parses an xml file: parseXML.html, parseXML.js, cities.xml

cities.xml

classes in JavaScript

“An object is an instance of a class”
This is the seminal definition in object oriented programming. In fact, a class defines every object in it, and the class ensures that every object has the same attributes and methods. However, JavaScript actually has no classes, which is to say that every object can be different and you are never sure whether an object has a particular attribute or method.

An object in JavaScript is just a variable with a bunch of attributes and methods attached to it. One way you can try to make a group of objects all have the same attributes is by creating each object in the group with the same function. This function is a called a “constructor”, and if it assigns an attribute to the new object it is constructing, then every object created with that constructor will have that attribute.

One way of ensuring that a group of objects all have the same methods defined on them is to define the methods using the word “prototype” in the method’s name definition. Notice the way the we call the constructor in the window.onload() function at the bottom of the program below. This way of calling the constructor ensures that the new Rectangle has the length and width attributes and any method whose name starts with “Rectangle.prototype”

For example, the following JavaScript program creates Rectangle objects that all have the same attributes (length and width) and methods (setRectangle() and showRectangle()):

```
// Demonstrates the definition and use of objects by defining Rectangle objects

// Constructor function, creates the two fields and initializes them to 0
Rectangle = function()
{   this.length = 0;          // fields inside every Rectangle object
    this.width = 0;
}

// Setter sets length to "l" and width to "w"
Rectangle.prototype.setRectangle = function(l, w)
{   this.length = l;
    this.width = w;
}

// Displays an alert box showing the length and width of the Rectangle
Rectangle.prototype.showRectangle = function()
{  var message = "length= " + this.length + "\nwidth= " + this.width;
   alert(message);
}

// Demonstrates two Rectangle objects with different values inside them.
window.onload = function()
{  var rectangle1 = new Rectangle();
   rectangle1.setRectangle(2,3);
   rectangle1.showRectangle();
   alert (rectangle1.length);     // all fields are public
   var rectangle2 = new Rectangle();
   rectangle2.setRectangle(5,6);
   rectangle2.showRectangle();
   rectangle1.showRectangle();    // rectangle1 still exists
}
```

class DropDown

The following section contains the source code for a class DropDown. Notice that this code can be used to create drop down menus of any sort, since the "class" definition doesn't mention courses or departments or cities or states.

I paste the "class" definition here, and you can download the whole program in the next section:

```
// A class that makes and updates a drop down menu containing "items" as the options,
// the menu is enclosed by "enclosingTagId" and has "callback" as its onchange event.
function DropDown(items, enclosingTagId, callBack)
{   this.selectNode = document.getElementById(enclosingTagId);
```

```
    this.selectNode.setAttribute("onchange", callBack);
    this.updateItems(items);
}

// After the drop down menu is there, this changes the options on it to contain items
DropDown.prototype.updateItems=function(items)
{   this.selectNode.options.length = 0;  // get rid of old options in the select node
    for ( i = 0; i < items.length; ++i)
    {   addNode (this.selectNode, "option", items[i], items[i]);
    }
}
```

create and use DropDown objects in this way:

```
var stateDropDown;
stateDropDown = new DropDown (stateList, "selectState", "makeCityMenu()");
var cityMenu;
cityMenu = new DropDown(stateObject.cities, "selectCity"); // no callback needed here
//if city menu already exists, we just have to update its options
cityMenu.updateItems(stateObject.cities);
```

DropDown.html

Instructions: The following files are needed to demonstrate the use of "class" DropDown: DropDown.html, DropDown.js, cities.json

DropDown.html

DropDown.js

Instructions: The following files are needed to demonstrate the use of "class" DropDown: DropDown.html, DropDown.js, cities.json

DropDown.js

cities.json

Instructions: The following files are needed to demonstrate the use of "class" DropDown: DropDown.html, DropDown.js, cities.json

cities.json

JSON

JSON stands for JavaScript Object Notation. This is a way of encoding data in the form of javascript object literals and array literals. This is usually a more efficient way of transferring data from the server to the client because it doesn't require conversion like XML does.

A json data file looks like a JavaScript variable declaration because that is exactly what it is. However, it not only declares the variable but also sets it equal to some literal value(s). For example, the following is in json notation:
var point = {x : 10, y: 20};

Here are the syntax rules for representing objects in JavaScript:
1) objects are surrounded by curly braces “{}”
2) attributes of objects and their corresponding values are separated by a colon “:”
3) the different attributes listed inside an object are separated by commas “,”
4) arrays are surrounded by square brackets “[]”
5) the different elements of an array are separated by commas “,”

Here is an example of a more complex json file:

```
// filename: cities.json
```

```
// Declares the USCities variable and fills it with data

var USCities =
{
  "states":
  [
    {   "stateName" : "California",
        "cities" : ["Alameda", "Albany", "Calaveras", "Angel's Camp"]
    },
    {   "stateName" : "Oregon",
        "cities" : ["Bend", "Portland", "Eugene"]
    },
    {   "stateName"  : "Washington",
        "cities" : ["Seattle", "Spokane", "Olympia", "Bellingham"]
    }
  ]
}
```

This data file can be treated as just another JavaScript source code file. If the data is static (known when the window is loaded), then the data file can just be named in an html script tag

jsonExample.html

Instructions: The following files are need to run the jsonExample: jsonExample.html, jsonExample.js, cities.xml (given earlier in this module)

jsonExample.html

jsonExample.js

Instructions: The following files are need to run the jsonExample: jsonExample.html, jsonExample.js, cities.xml (given earlier in this module)

jsonExample.js

The Use of eval()

This simple little function **eval()** can be very confusing. I include here three examples of its use.

1) If you have code in a string, you must use eval() to get JS to evaluate it as code. Note the difference between these two:

```
alert ( '7*5' );
alert ( eval('7*5') );
```

2) If you are looking for an attribute that comes into a function as a String parameter, you must evaluate the string parameter before evaluating the expression for the variable.

For example, what if you need to find the value stored in this variable?

```
USCities.states[0].cities[1]
```

The expression above works only if cities is the actual name of the attribute, which it is. But what if the name of the attribute is in a variable, as it would be if it were a parameter passed to a function? In this case, you need to evaluate the parameter first to see what value is in there, before you are able to access the variable itself. See the use of eval in this function:

```
// Looks through an array of objects to find the object with
// a particular attribute/value pair. Returns the object.
function findObject (array, attribute, value)
{  var targetObject = null;
   // find the object in array with attribute == value
   for (i = 0; i < array.length && targetObject==null; ++i)
   {  if (eval("array[" + i + "]."+attribute) == value)
```

```
        {   targetObject = array[i];
        }
    }
    return targetObject;
}
```

Use function findObject() in this way:

```
var stateObject = findObject(USCities.states, “stateName”, “Oregon”)
alert(stateObject.cities);   // will give the cities in Oregon
```

3) Server side software can be written to generate data in JSON format. In order to get the server to give you only the data you want, your program can put specific information in the XMLHttpRequest object to specify which data you want. This method is in contrast to putting static data in a file sitting next to your JavaScript source file, which you access with an html script tag