

Homework: XML Exercise

1. Objectives

- Become familiar with the DOM paradigm;
- Use an existing XML parser;
- Transform the content of an XML document into an HTML page.

2. Description

You are required to write an HTML/JavaScript program, which takes the URL of an XML document containing the final results of selected sports at the 2008 Olympic Games, parses the XML file, and displays results in an HTML document. The resulting HTML document will display a selection list consisting of the names of the Sports and a “Show Support Events” button. Clicking on the button will display another selection list consisting of the names of different events for the Sport selected in the first selection list and a “Show Event Winner” button. Clicking on this button will display information about the winner of the event in an HTML table along with a picture and a video related to the event or the athlete from youtube.com. The JavaScript program will be implemented by embedding it in a HTML file, so that it can be executed within a browser.

- Initially your program should display a single text box that allows the entering of an arbitrary URL of an XML document containing selected results of the 2008 Olympic Games. The interface should look like the following Figure 1:

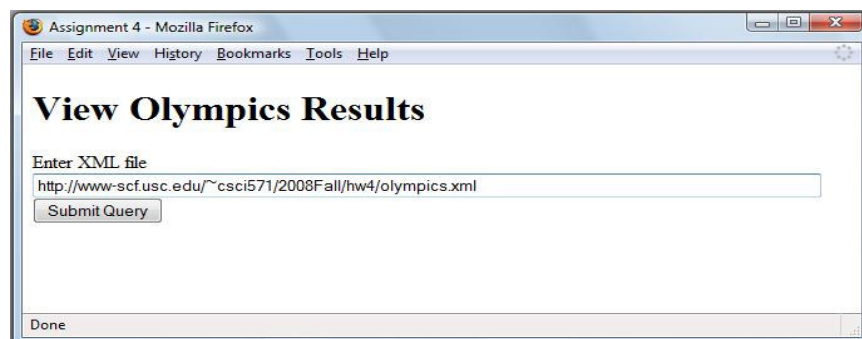


Figure 1. Initial web page

You should implement this form using only HTML.

- The form, when the “Submit Query” button is clicked, would call an HTML/JavaScript function within the HTML file that parses the XML document given as an input.
- After parsing the XML document, a new popup window should be displayed consisting of a drop down selection list that should contain one entry for each

Sport and a “Show Sport Events” button next to it, as shown in Figure 2. For example, given the following XML document:

<http://www-scf.usc.edu/~csci571/2008Fall/hw4/olympics.xml>

your program should produce the web page shown in Figure 2 below.

- After selecting a Sport and clicking the “Show Sport Events” button, another selection list consisting of the events related to the Sport selected in first selection list and a “Show Event Winner” button should appear as shown in Figure 3.
- After selecting an event and clicking the “Show Event Winner” button, a table should be displayed containing the details of the event winner including the name of the event winner, winner’s country, the final timing of the event, the picture of the athlete and a youtube.com video embedded in the last row of the table as shown in Figure 4. The data shown in the table cells and the URLs for the pictures and the videos are to be read from the XML document. You will have to embed the video in the HTML page. See Section 3, Step 5 for details about embedding a YouTube video in a webpage.

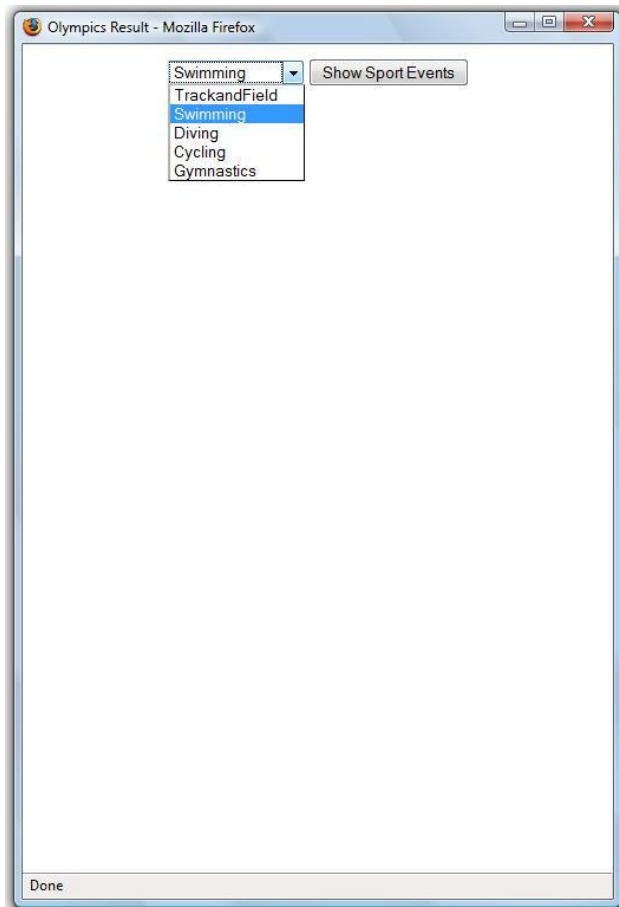


Figure 2. Selection list and “Show Sport Events” button generated after processing olympics.xml

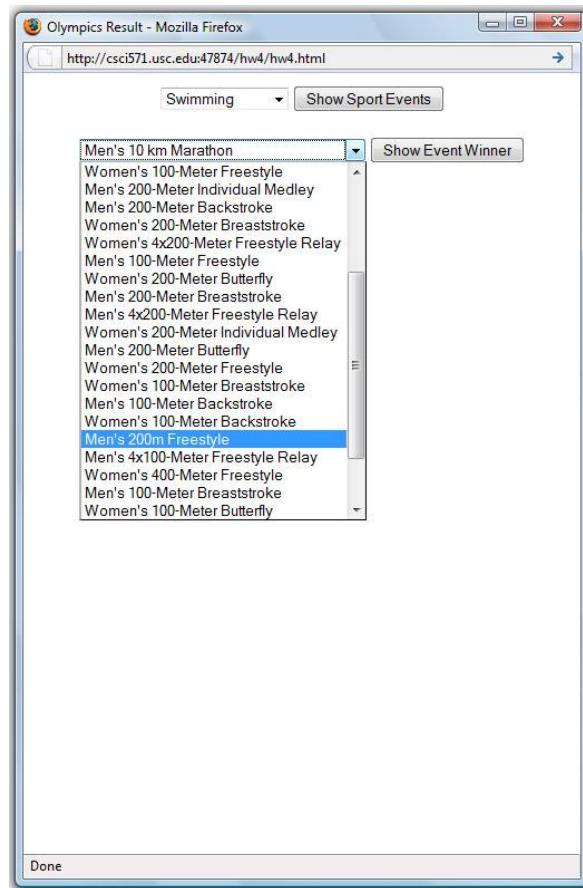


Figure 3. Selection list and “Show Event Winner” button generated after clicking “Show Sport Events”

- If a value is changed in the first selection list that contains the Sports names, all the elements on the page showing under the first selection list and the “Show Sport Events” button should be cleared. After the “Show Sport Events” button is clicked, the events selection list and the “Show Event Winner” button should appear and then on clicking the “Show Event Winner” button, the table should be displayed.
- Similarly if the value in the second selection list containing the event names is changed, the table containing the details for the previous events should be cleared. After the “Show Event Winner” button is clicked, the table should appear again showing the details of the newly selected event.
- You should use the XML DOM Parser that comes as a built-in component in both IE and Firefox to parse and validate the XML document. Your code can assume that the XML files your program will process will be structured as a “three level tree” as described in Section 3, Step 1, starting on page 6.

- If the XML file is invalid (e.g. missing the closing tag or having tags that overlap), an error message should be generated in a popup window as shown in Figure 5.

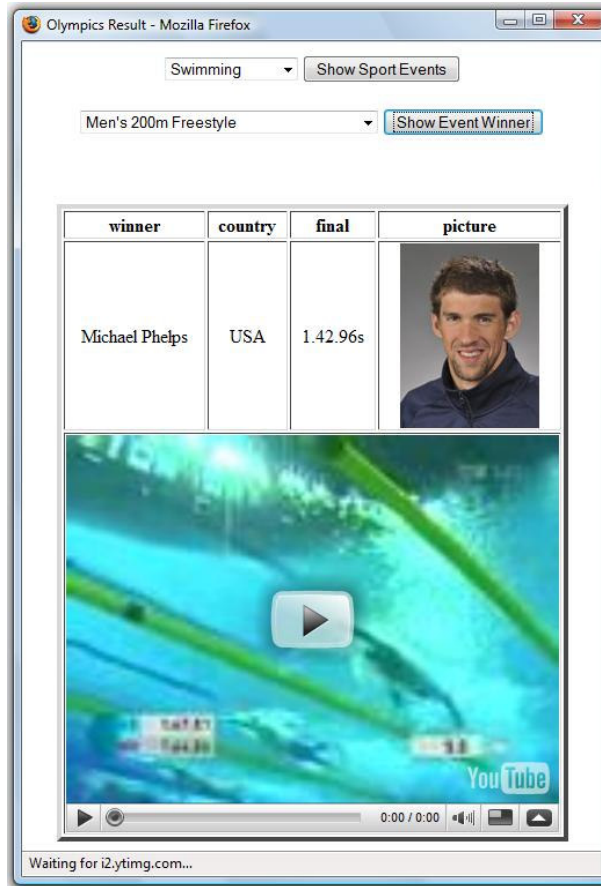


Figure 4. Table containing event winner details shown after clicking “Show Event” Winner button

In case of a parsing error, your program should pop up the following web page:

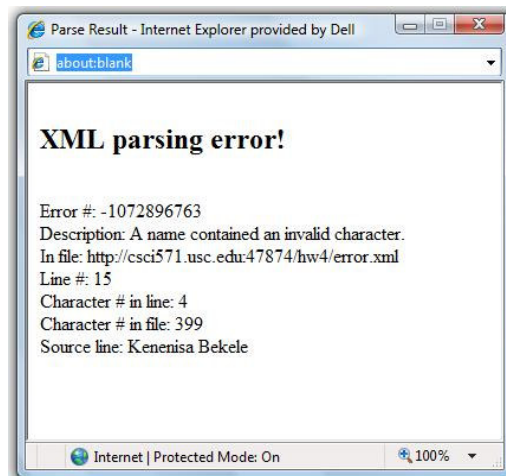


Figure 5. Page generated from error.xml

For example, the error above is displayed when parsing the file:

`http://www-scf.usc.edu/~csci571/2008Fall/hw4/error.xml`

Since and Firefox DOM implementations do not contain the `parseError` object, you should show an alert box if the XML file is not valid, as shown in the following Figure 6:

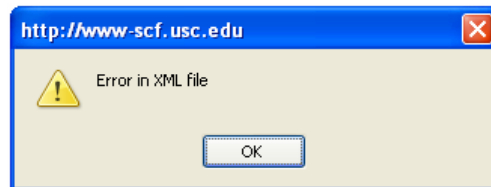


Figure 6. Error generated from error.xml in Firefox

3. Hints

- *Step 1: Writing Your HTML/JavaScript program - Using the DOM Parser*

IE and Firefox have a built-in DOM parser. Microsoft DOM implementation is part of Microsoft XML Core Services (MSXML). An introduction to the Microsoft DOM can be found at:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk/html/a09c098b-7e5a-45ff-b5ad-bc910f736a3f.asp>

The set of DOM enumerated constants can be found at:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk/html/e94d1cdf-3d7f-4ac0-8c51-a17d2801519c.asp>

Please note that since the MS XML parser does not define the DOM constants as in `Node.ELEMENT_NODE`, you will have to define them in your code, as in:

```
ELEMENT_NODE = 1;
```

The set of DOM properties, methods and events can be found at:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk/html/d051f7c5-e882-42e8-a5b6-d1ce67af275c.asp>

Here's how you could use the Microsoft DOM API and the Mozilla DOM API used in Firefox to load and parse an XML document into a DOM tree, and then use the DOM API to extract information from that document.

```

<script LANGUAGE=JavaScript>
//create an instance of the XML parser
if (window.ActiveXObject){ //Checking if the browser is IE
    var xmlDoc = new ActiveXObject("Microsoft.XMLDOM")
    xmlDoc.async="false" //make sure doc is fully loaded
    xmlDoc.load(filename) //load the file in the parser
    if (xmlDoc.parseError.errorCode != 0) {
        var myError = xmlDoc.parseError;
        alert("You have error " + myError.reason);
    } else {
        // ..... processing the document goes here
        alert(xmlDoc.xml);
    }
}
}

else if (document.implementation && document.implementation.createDocument)//for
mozilla based browsers
{
    var xmlDoc= document.implementation.createDocument("", "doc", null); // NS
    xmlDoc.async=false; //make sure doc is fully loaded
    loaded = xmlDoc.load(URL);
    if(!loaded){
        alert("Error");
    }
    else
    {
        alert(xmlDoc.xml);
    }
}
}
</script>

```

Now you can generate the HTML table from the DOM tree. You must make sure to write your program so it does not explicitly make use of the tag names in the sample XML file mentioned above. Instead, your program should assume that the XML files that your program will process will be structured as a three level tree with the following format:

```

<Results>
  <Result>
    <name>Track and Field</name>
    <event>
      <name> . . . </name>
      <winner> . . . </winner>
      <country> . . . </country>
      <final>. . . </final>
      <picture>. . . </picture>
      <video>. . . </video>
    </event>
    . . .
    <event>
      <name> . . . </name>
      <winner> . . . </winner>
      <country> . . . </country>
      <final>. . . </final>
      <picture>. . . </picture>
      <video>. . . </video>
    </event>
  </Result>
  <Result>
    <name>Swimming</name>

```

```

    <event>
        <name> . . . </name>
        <winner> . . . </winner>
        <country> . . . </country>
        <final>. . . </final>
        <picture>. . . </picture>
        <video>. . . </video>
    </event>
    . . .
    <event>
        <name> . . . </name>
        <winner> . . . </winner>
        <country> . . . </country>
        <final>. . . </final>
        <picture>. . . </picture>
        <video>. . . </video>
    </event>
</Result>
. . .
</Results>

```

Your task is to write a program that transforms this structure into HTML, where first level tags contain the results for each sport and the first tag at the second level for each Result will have the name of the Sport in the tag name, e.g.<name>Swimming</name>. All the tags appearing at the second level <name>,<event> and the six tags <name>, <winner>, <country>, <final>, <picture> and <video> appearing at the third level are fixed and your program may assume that in a certain XML document, the entries at level 2 and 3 are always in the same order, and that no entries are missing. However, you do not know how many records of Result and events may occur in the file. When producing the table, your program should use the tag names “name”, “winner”, “country”, “final” and “picture” as the table headers as shown in Figure 3. Please note, the headers should be dynamically populated from the XML file tags and not from some constant strings written in the code. Please also note that after clicking the buttons or selecting some different value in the selection lists, the data in the Selection list and the table should be updated dynamically and no page refresh should occur.

You can access the child nodes of the documents as follows:

```

mynodes=xmlDoc.documentElement.childNodes;

```

Note that unlike the Java-based DOM, which provides methods such as `getChildNodes()` and `getNodeTypes()` that return a node list of children of a current node and the type of a node respectively, with the DOM you have to access the element properties directly, as in:

```

myNodeList= mynodes.item(i).childNodes;
if (myNodeList.item(j).nodeType==ELEMENT_NODE)

```

- *Step 2: Display the Resulting HTML Document*

You should use the DOM `document.write` method to produce the required HTML, as in:

```
html_text="<html><head><title>XML Result</title></head><body>";
hWin.document.write(html_text);
```

- *Step 3: Use JavaScript control syntax*

The only program control statements that you will need to use for this exercise are the “if”, the “for” and some basic string manipulation statements. The syntax of both “if” and “for” statements is practically identical to the syntax of the corresponding statement in the C, C++ and Java languages, as in:

```
if(xmlDoc.documentElement.hasChildNodes()) {
    // do stuff
}
for (j=0;j<albumNodeList.length;j++) {
    // do more stuff
}
```

- *Step 4: Use JavaScript Object API to open new Windows*

The popup window that you should display containing an HTML table or a parsing error (see section 2, “Description”) should be created using the JavaScript `window.open` object API, as in:

```
hWin=window.open("", "Assignment", "height=700,width=500");
// write to the window
hWin.document.write(html_text);
```

The full syntax and long list of “features” of `window.open` can be found at:

<http://developer.mozilla.org/en/docs/DOM:window.open>

- *Embedding a YouTube video in the html document*

To show a YouTube video, the webpage should contain the following embedding code:

```
<object width="425" height="344"><param name="movie"
value="URL"></param><param name="allowFullScreen"
value="true"></param><embed src="URL" type="application/x-
shockwave-flash" allowfullscreen="true" width="425"
height="344"></embed></object>
```

where URL in both value and src values should be replaced by the URL contained in the `<video>` tag of the XML document.

- *Handling of White Spaces in DOM in Firefox*

Note, that white spaces are handled differently in Firefox as compared to IE. Below is the link to the web page that demonstrates how to handle white spaces in Mozilla:

http://developer.mozilla.org/en/docs/Whitespace_in_the_DOM

4. Material You Need to Submit

Complete URLs to both `olympics.xml` and `error.xml` are given in Section 2. On your course homework page, your link for this homework should go to a page that looks like the one displayed in Figure 1. This page should include your entire JavaScript/HTML program in a single file. Also you should ‘submit’ your source code and a README file electronically to the csci571 account, so that it can be graded and compared to all other students' code via the MOSS code comparison tool.