Project #2 – XML Document Model

due Thursday, Mar 19

Version 1.2

Purpose:

This project requires you to develop an XML parsing facility that reads XML strings or text files, builds a Document object that can be queried for information about the XML contents, supports programmatic modification of the Document, and can write the revisions to another XML string or file. The project requires you to develop C++ packages to: parse the input, build a tree-based in-memory representation of the XML called a parse tree[1], and support modifications of that representation. This facility will support read and write operations. It will also support the programmatic creation of new XML strings and files.

Requirements:

Your XML Document project:

1. **Shall** use standard C++[2] and the standard library, compile and link from the command line, using Visual Studio 2013, as provided in the ECS clusters and operate in the environment provided there[3].

2. **Shall** use services of the C++ std::iostream library for all input and output to and from the user's console and C++ operator new and delete for all dynamic memory management. You are encouraged use one of the standard C++11 smart pointer types to support memory management instead of directly managing heap allocations.

3. **Shall** provide a facility to read XML strings and files and build an internal parse tree representation wrapped in a Document object. Each XML element is represented by a node in the tree. Some elements may have a finite number of attributes, e.g., name-value pairs, stored in a std::vector in the element node. Also, some elements may have child elements.

4. The Document **shall** support move, and move assignment operations[4] as well as read and write operations to and from both strings and files.

5. **Shall** provide the capability to find any element based on a unique id attribute[5] for all those elements that possess id attributes. If an element with the specified id attribute is found, a pointer to the element node is returned. If no such element exists a null pointer is returned.

6. **Shall** provide the capability to find a collection of elements that have a specified tag. The elements are returned with a std::vector that holds pointers to each element that has the specified tag. If no such elements exist, an empty vector is returned.

7. **Shall** provide the capability to add a child element to, and remove a child from, any element in the tree that can hold child references, found by id or tag. Addition returns a boolean success value. **Shall** also provide the ability to add a root element to an empty document tree[6]. **Shall** provide the capability to remove a child from any element in the tree that can hold child references, found by id or tag. Removal returns a boolean success value.

8. **Shall**, given a pointer to any element, provide a facility to return a std::vector containing all the name-value attribute pairs attached to that element. If the element has no attributes an empty std::vector is returned. **Shall** also provide a facility to return a std::vector of pointers to all the children of a specified element. If the element has no children an empty std::vector is returned.

9. **Shall** provide the ability to add or remove an attribute name-value pair from any element node that supports attributes.

---

[1] Parse trees are not required to support balancing operations since XML is not inherently balanced.
[2] This means, for example that you may not use the .Net managed extensions to C++.
[3] VC++ version 12.0 is provided by Visual Studio 2013, and is available in all the ECS clusters.
[4] The move and move assignment operations are relatively easy to implement while copy and copy assignment for a linked data structure like an XmlDocument is difficult. It is recommended that you disallow copy construction and copy assignment while supporting move construction and move assignment.
[5] This is intended to work something like Microsoft .Net's XmlDocument.GetElementbyId(string) where the input is an attribute **value** and the result is an XmlElement.
[6] The intent is to support construction of an XmlDocument programmatically, without starting by reading XML text from a file or string.

10. **Shall** provide the capability to read an XML string or file and build the corresponding internal tree representation. Shall also provide the capability to write an XML string or file corresponding to the internal tree representation.

11. **Shall** provide a test executive package that accepts a command line argument specifying an XML file to parse, displays the corresponding tree structure, and continues on to demonstrate each of the functional requirements given here[7].

12. Your project submission **shall** be uploaded in a zip file archive, including two batch files named compile.bat and run.bat that compile your project and run it using appropriate command line arguments. Please also include a Visual Studio solution that when run demonstrates you meet these requirements[8].

Note that there is no requirement to provide a graphical user interface. If you do so, you should also provide a command line interface, as required in #11. Please demonstrate that you meet all of the requirements, stated above.

[7] This requirement asks you to demonstrate reading XML from a string as well as from a file.
[8] This implies that you are required to demonstrate programmatically that you meet all these requirements. You can do that by having your executive execute a series of steps, each of which shows that you meet a requirement and the entire sequence demonstrates you meet all of them.