XML Parser: Parsing an XML CD Catalog

John Businge

January 17, 2023

Extensible Markup Language (XML) is a markup language and file format for storing, transmitting, and reconstructing arbitrary data. It defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

There are several implementations for an XML parser in C++. In principle you may use a parser of your choice. In this lab we will learn to work with the "TinyXML" library.

In this first lab is prepared for that you learn to work with this XML parser so you can use it in the project.

TinyXML Tutorial

XML is a relatively simple data format. It is human readable and very similar to HTML. Below is a general example of a correct XML file:

```
<?xml version="1.0" ?>
<root>
    <Element1 attribute1="some value" />
    <Element2 attribute2="2" attribute3="3">
        <Element3 attribute4="4" />
        Some text.
    </Element2>
</root>
```

Download the XML Parser here: http://sourceforge.net/projects/tinyxml/ Place these files in your project:

```
tinystr.cpp
tinyxmlerror.cpp
tinystr.h
tinyxml.h
tinyxml.cpp
tinyxmlparser.cpp
```

Don't forget to include these files in the `CMakeLists.txt` too! To use the `TinyXML` library we need to include "`tinyxml.h`" in our code:

```
#include "tinyxml.h"
```

Now we can load a document. Create a file with the XML sample from above and name it "`test.xml`". We can now load this file with:

```
TiXmlDocument doc;
if(!doc.LoadFile("test.xml")) {
    std::cerr << doc.ErrorDesc() << std::endl;
    return 1;
}
```

Note: By default, `CLion` runs the application from the `cmake-build-debug` folder and will also search for files there. As a result, the `test.xml` file will not be found. You can edit configurations. . . modify the working directory to solve this.

The variable doc now contains all data. We can extract the data like this:

```
TiXmlElement* root = doc.FirstChildElement();
if(root == NULL) {
    std::cerr << "Failed to load file: No root element." << std::endl;
    doc.Clear();
    return 1;
}
```

We now have a variable that contains the root element. We can access the rest of the data via this element in the same way as in the previous code: the `FirstChildElement()` method returns a `TiXmlElement` pointer pointing to the first child node. Every class derived from `TiXmlNode` (including `TiXmlDocument` and `TiXmlElement`) contains this method.

The FirstChildElement() method takes a string with the name of the element you are looking for as an optional argument. We didn't need these because we knew that there was only one root. However, we now know that our root has two children with specific name. So we could use the FirstChildElement() method with the name of each element. But, because the number of nodes and their name is not always known, we will use more general code:

```
for(TiXmlElement* elem = root->FirstChildElement(); elem != NULL;
                         elem = elem->NextSiblingElement()) {
    string elemName = elem->Value();
```

This code walks over the elements that are direct children of `root`. We specify no name in the calls to `FirstChildElement()` and `NextSiblingElement()`. Therefore we need to check what the name is. The `Value()` method is different for each class derived from `TiXmlNode`: for `TiXmlElement` it returns the name of the element back.

```
       const char* attr;
       if(elemName == "Element1") {
              attr = elem->Attribute("attribute1");
              if(attr != NULL)
                  ; // Do stuff with it
       }
```

The variable `attr` is used to keep track of the requested attributes. If the requested attribute does not exist, the `Attribute()` method will return `NULL`.

We can query Element2's attributes in the same way:

```
else if(elemName == "Element2") {
    attr = elem->Attribute("attribute2");
    if(attr != NULL)
        ; // Do stuff with it
attr = elem->Attribute("attribute3");
if(attr != NULL)
    ; // Do stuff with it
```

Element2 also contains a child: Element3. We will find Element3 with a loop to show how to use a more specific loop. This run will skip all elements that are not named 'Element3'.

```
for(TiXmlElement* e = elem->FirstChildElement("Element3"); e != NULL;
                                e = e->NextSiblingElement("Element3")){
    attr = e->Attribute("attribute4");
    if(attr != NULL)
    ; // Do stuff with it
}
```

Attributes are not the only way to keep track of data in XML. Another common way is the text contained within an element, such as in element2. `TinyXML` keeps this text in a text node (TiXmlText). Since there is no `FirstChildText()` method, we use the `ToText()` method to cast the `TiXmlNode` to a `TiXmlText` class. If it is `NULL`, it has no text node. We can get the text with the `Value()` method from `TiXmlText`.

```
    for(TiXmlNode* e = elem->FirstChild(); e != NULL; e = e->NextSibling()){
        TiXmlText* text = e->ToText();
        if(text == NULL)
            continue;
        string t = text->Value();
        // Do stuff
    }
```

This parsed our entire example XML file. We can now remember that `TinyXML` uses now release as follows:

```
doc.Clear();
```

Exercise 1: Reading and printing data

The file eenCD.xml contains an XML description of a CD. Write a module (Main) which parses a eenCD.xml file. Next, in this module you have to: Retrieve read-in Artist and Title and print it in the console output.

Exercise 2: More OO Focused

a) Create a new module in which you define an Object CD. This item keeps track of the `Artist`, `Title`, `Year` and `Price` of the CD. Provide `getter` and `setter` procedures to modify the data.

b) Now make sure that an object CD is created with the information that was parsed. Provide a printing method so that the Artist, Title, Price and Year are printed can become.

(!!! You will need to convert string (`char *`) objects to `doubles` and `ints`. Look at the internet for a suitable method.)

Exercise 3: From XML to ADT

In this exercise we will extend the previous exercise so that we now have a list of Read CDs. The `cdCatalog.xml` file contains a list of CDs. You may use a vector object from the standard library (`std`).

(!!! Attention: the data with Tag "CD" is now a nesting level deeper than in the previous exercise)

The Main module should now be modified so that it displays the list of CDs on the output performs.