TinyXml Documentation

2.6.2

TinyXML

TinyXML is a simple, small, C++ XML parser that can be easily integrated into other programs.

What it does.

In brief, TinyXML parses an XML document, and builds from that a Document Object Model (DOM) that can be read, modified, and saved.

XML stands for "eXtensible Markup Language." It allows you to create your own document markups. Where HTML does a very good job of marking documents for browsers, XML allows you to define any kind of document markup, for example a document that describes a "to do" list for an organizer application. XML is a very structured and convenient format. All those random file formats created to store application data can all be replaced with XML. One parser for everything.

The best place for the complete, correct, and quite frankly hard to read spec is at http://www.w3.org/TR/2004/REC-xml-20040204/. An intro to XML (that I really like) can be found at http://skew.org/xml/tutorial.

There are different ways to access and interact with XML data. TinyXML uses a Document Object Model (DOM), meaning the XML data is parsed into a C++ objects that can be browsed and manipulated, and then written to disk or another output stream. You can also construct an XML document from scratch with C++ objects and write this to disk or another output stream.

TinyXML is designed to be easy and fast to learn. It is two headers and four cpp files. Simply add these to your project and off you go. There is an example file - xmltest.cpp - to get you started.

TinyXML is released under the ZLib license, so you can use it in open source or commercial code. The details of the license are at the top of every source file.

TinyXML attempts to be a flexible parser, but with truly correct and compliant XML output. TinyXML should compile on any reasonably C++ compliant system. It does not rely on exceptions or RTTI. It can be compiled with or without STL support. TinyXML fully supports the UTF-8 encoding, and the first 64k character entities.

What it doesn't do.

TinyXML doesn't parse or use DTDs (Document Type Definitions) or XSLs (eXtensible Stylesheet Language.) There are other parsers out there (check out www.sourceforge.org, search for XML) that are much more fully featured. But they are also much bigger, take longer to set up in your project, have a higher learning curve, and often have a more restrictive license. If you are working with browsers or have more complete XML needs, TinyXML is not the parser for you.

The following DTD syntax will not parse at this time in TinyXML:

```
<!DOCTYPE Archiv [
 <!ELEMENT Comment (#PCDATA)>
]>
```

because TinyXML sees this as a !DOCTYPE node with an illegally embedded !ELEMENT node. This may be addressed in the future.

Tutorials.

For the impatient, here is a tutorial to get you going. A great way to get started, but it is worth your time to read this (very short) manual completely.

- **TinyXML Tutorial**

Code Status.

TinyXML is mature, tested code. It is very stable. If you find bugs, please file a bug report on the sourceforge web site (www.sourceforge.net/projects/tinyxml). We'll get them straightened out as soon as possible.

There are some areas of improvement; please check sourceforge if you are interested in working on TinyXML.

Related Projects

TinyXML projects you may find useful! (Descriptions provided by the projects.)

- **TinyXPath** (http://tinyxpath.sourceforge.net). TinyXPath is a small footprint XPath syntax decoder, written in C++.
- **TinyXML++** (http://code.google.com/p/ticpp/). TinyXML++ is a completely new interface to TinyXML that uses MANY of the C++ strengths. Templates, exceptions, and much better error handling.

Features

Using STL

TinyXML can be compiled to use or not use STL. When using STL, TinyXML uses the std::string class, and fully supports std::istream, std::ostream, operator<<, and operator>>. Many API methods have both 'const char*' and 'const std::string&' forms.

When STL support is compiled out, no STL files are included whatsoever. All the string classes are implemented by TinyXML itself. API methods all use the 'const char*' form for input.

Use the compile time define:

TIXML_USE_STL

to compile one version or the other. This can be passed by the compiler, or set as the first line of "tinyxml.h".

Note: If compiling the test code in Linux, setting the environment variable TINYXML_USE_STL=YES/NO will control STL compilation. In the Windows project file, STL and non STL targets are provided. In your project, It's probably easiest to add the line "#define TIXML_USE_STL" as the first line of **tinyxml.h**.

UTF-8

TinyXML supports UTF-8 allowing to manipulate XML files in any language. TinyXML also supports "legacy mode" - the encoding used before UTF-8 support and probably best described as "extended ascii".

Normally, TinyXML will try to detect the correct encoding and use it. However, by setting the value of TIXML_DEFAULT_ENCODING in the header file, TinyXML can be forced to always use one encoding.

TinyXML will assume Legacy Mode until one of the following occurs:

1. If the non-standard but common "UTF-8 lead bytes" (0xef 0xbb 0xbf) begin the file or data stream, TinyXML will read it as UTF-8.
2. If the declaration tag is read, and it has an encoding="UTF-8", then TinyXML will read it as UTF-8.
3. If the declaration tag is read, and it has no encoding specified, then TinyXML will read it as UTF-8.
4. If the declaration tag is read, and it has an encoding="something else", then TinyXML will read it as Legacy Mode. In legacy mode, TinyXML will work as it did before. It's not clear what that mode does exactly, but old content should keep working.
5. Until one of the above criteria is met, TinyXML runs in Legacy Mode.

What happens if the encoding is incorrectly set or detected? TinyXML will try to read and pass through text seen as improperly encoded. You may get some strange results or mangled characters. You may want to force TinyXML to the correct mode.

You may force TinyXML to Legacy Mode by using LoadFile(TIXML_ENCODING_LEGACY) or LoadFile(filename, TIXML_ENCODING_LEGACY). You may force it to use legacy mode all the time by setting TIXML_DEFAULT_ENCODING = TIXML_ENCODING_LEGACY. Likewise, you may force it to TIXML_ENCODING_UTF8 with the same technique.

For English users, using English XML, UTF-8 is the same as low-ASCII. You don't need to be aware of UTF-8 or change your code in any way. You can think of UTF-8 as a "superset" of ASCII.

UTF-8 is not a double byte format - but it is a standard encoding of Unicode! TinyXML does not use or directly support wchar, TCHAR, or Microsoft's _UNICODE at this time. It is common to see the term "Unicode" improperly refer to UTF-16, a wide byte encoding of unicode. This is a source of confusion.

For "high-ascii" languages - everything not English, pretty much - TinyXML can handle all languages, at the same time, as long as the XML is encoded in UTF-8. That can be a little tricky, older programs and operating systems tend to use the "default" or "traditional" code page. Many apps (and almost all modern ones) can output UTF-8, but older or stubborn (or just broken) ones still output text in the default code page.

For example, Japanese systems traditionally use SHIFT-JIS encoding. Text encoded as SHIFT-JIS can not be read by TinyXML. A good text editor can import SHIFT-JIS and then save as UTF-8.

The Skew.org link does a great job covering the encoding issue.

The test file "utf8test.xml" is an XML containing English, Spanish, Russian, and Simplified Chinese. (Hopefully they are translated correctly). The file "utf8test.gif" is a screen capture of the XML file, rendered in IE. Note that if you don't have the correct fonts (Simplified Chinese or Russian) on your system, you won't see output that matches the GIF file even if you can parse it correctly. Also note that (at least on my Windows machine) console output is in a Western code page, so that Print() or printf() cannot correctly display the file. This is not a bug in TinyXML - just an OS issue. No data is lost or destroyed by TinyXML. The console just doesn't render UTF-8.

Entities

TinyXML recognizes the pre-defined "character entities", meaning special characters. Namely:

```
&amp;   &
&lt;    <
&gt;    >
&quot;  "
&apos;  '
```

These are recognized when the XML document is read, and translated to there UTF-8 equivalents. For instance, text with the XML of:

```
Far &amp; Away
```

will have the Value() of "Far & Away" when queried from the **TiXmlText** object, and will be written back to the XML stream/file as an ampersand. Older versions of TinyXML "preserved" character entities, but the newer versions will translate them into characters.

Additionally, any character can be specified by its Unicode code point: The syntax " " or " " are both to the non-breaking space characher.

Printing

TinyXML can print output in several different ways that all have strengths and limitations.

- Print(FILE*). Output to a std-C stream, which includes all C files as well as stdout.
  - "Pretty prints", but you don't have control over printing options.
  - The output is streamed directly to the FILE object, so there is no memory overhead in the TinyXML code.
  - used by Print() and SaveFile()

- operator<<. Output to a c++ stream.
  - Integrates with standart C++ iostreams.
  - Outputs in "network printing" mode without line breaks. Good for network transmission and moving XML between C++ objects, but hard for a human to read.

- **TiXmlPrinter**. Output to a std::string or memory buffer.
  - API is less concise
  - Future printing options will be put here.
  - Printing may change slightly in future versions as it is refined and expanded.

Streams

With TIXML_USE_STL on TinyXML supports C++ streams (operator <<,>>) streams as well as C (FILE*) streams. There are some differences that you may need to be aware of.

C style output:

- based on FILE*
- the Print() and SaveFile() methods

Generates formatted output, with plenty of white space, intended to be as human-readable as possible. They are very fast, and tolerant of ill formed XML documents. For example, an XML document that contains 2 root elements and 2 declarations, will still print.

C style input:

- based on FILE*
- the Parse() and LoadFile() methods

A fast, tolerant read. Use whenever you don't need the C++ streams.

C++ style output:

- based on std::ostream
- operator<<

Generates condensed output, intended for network transmission rather than readability. Depending on your system's implementation of the ostream class, these may be somewhat slower. (Or may not.) Not tolerant of ill formed XML: a document should contain the correct one root element. Additional root level elements will not be streamed out.

C++ style input:

- based on std::istream
- operator>>

Reads XML from a stream, making it useful for network transmission. The tricky part is knowing when the XML document is complete, since there will almost certainly be other data in the stream. TinyXML will assume the XML data is complete after it reads the root element. Put another way, documents that are ill-constructed with more than one root element will not read correctly. Also note that operator>> is somewhat slower than Parse, due to both implementation of the STL and limitations of TinyXML.

White space

The world simply does not agree on whether white space should be kept, or condensed. For example, pretend the '_' is a space, and look at "Hello_____world". HTML, and at least some XML parsers, will interpret this as "Hello_world". They condense white space. Some XML parsers do not, and will leave it as "Hello_____world". (Remember to keep pretending the _ is a space.) Others suggest that __Hello___world__ should become Hello___world.

It's an issue that hasn't been resolved to my satisfaction. TinyXML supports the first 2 approaches. Call **TiXmlBase::SetCondenseWhiteSpace(bool)** to set the desired behavior. The default is to condense white space.

If you change the default, you should call **TiXmlBase::SetCondenseWhiteSpace(bool)** before making any calls to Parse XML data, and I don't recommend changing it after it has been set.

Handles

Where browsing an XML document in a robust way, it is important to check for null returns from method calls. An error safe implementation can generate a lot of code like:

```
TiXmlElement* root = document.FirstChildElement( "Document" );
if ( root )
{
        TiXmlElement* element = root->FirstChildElement( "Element" );
        if ( element )
        {
                TiXmlElement* child = element->FirstChildElement( "Child" );
                if ( child )
```

```
		{
			TiXmlElement* child2 = child->NextSiblingElement( "Child" );
			if ( child2 )
			{
				// Finally do something useful.
```

Handles have been introduced to clean this up. Using the **TiXmlHandle** class, the previous code reduces to:

```
TiXmlHandle docHandle( &document );
TiXmlElement* child2 = docHandle.FirstChild( "Document" ).FirstChild( "Element" ).Child( "Ch
ild", 1 ).ToElement();
if ( child2 )
{
	// do something useful
```

Which is much easier to deal with. See **TiXmlHandle** for more information.

Row and Column tracking

Being able to track nodes and attributes back to their origin location in source files can be very important for some applications. Additionally, knowing where parsing errors occured in the original source can be very time saving.

TinyXML can tracks the row and column origin of all nodes and attributes in a text file. The **TiXmlBase::Row()** and **TiXmlBase::Column()** methods return the origin of the node in the source text. The correct tabs can be configured in **TiXmlDocument::SetTabSize()**.

Using and Installing

To Compile and Run xmltest:

A Linux Makefile and a Windows Visual C++ .dsw file is provided. Simply compile and run. It will write the file demotest.xml to your disk and generate output on the screen. It also tests walking the DOM by printing out the number of nodes found using different techniques.

The Linux makefile is very generic and runs on many systems - it is currently tested on mingw and MacOSX. You do not need to run 'make depend'. The dependecies have been hard coded.

Windows project file for VC6

- tinyxml: tinyxml library, non-STL
- tinyxmlSTL: tinyxml library, STL
- tinyXmlTest: test app, non-STL
- tinyXmlTestSTL: test app, STL

Makefile

At the top of the makefile you can set:

PROFILE, DEBUG, and TINYXML_USE_STL. Details (such that they are) are in the makefile.

In the tinyxml directory, type "make clean" then "make". The executable file 'xmltest' will be created.

To Use in an Application:

Add tinyxml.cpp, **tinyxml.h**, tinyxmlerror.cpp, tinyxmlparser.cpp, tinystr.cpp, and **tinystr.h** to your project or make file. That's it! It should compile on any reasonably compliant C++ system. You do not need to enable exceptions or RTTI for TinyXML.

How TinyXML works.

An example is probably the best way to go. Take:

```
	<?xml version="1.0" standalone=no>
	<!-- Our to do list data -->
	<ToDo>
```

```
                <Item priority="1"> Go to the <bold>Toy store!</bold></Item>
                <Item priority="2"> Do bills</Item>
        </ToDo>
```

Its not much of a To Do list, but it will do. To read this file (say "demo.xml") you would create a document, and parse it in:

```
        TiXmlDocument doc( "demo.xml" );
        doc.LoadFile();
```

And its ready to go. Now lets look at some lines and how they relate to the DOM.

```
<?xml version="1.0" standalone=no>
```

The first line is a declaration, and gets turned into the **TiXmlDeclaration** class. It will be the first child of the document node.

This is the only directive/special tag parsed by TinyXML. Generally directive tags are stored in **TiXmlUnknown** so the commands wont be lost when it is saved back to disk.

```
<!-- Our to do list data -->
```

A comment. Will become a **TiXmlComment** object.

```
<ToDo>
```

The "ToDo" tag defines a **TiXmlElement** object. This one does not have any attributes, but does contain 2 other elements.

```
<Item priority="1">
```

Creates another **TiXmlElement** which is a child of the "ToDo" element. This element has 1 attribute, with the name "priority" and the value "1".

```
Go to the
```

A **TiXmlText**. This is a leaf node and cannot contain other nodes. It is a child of the "Item" **TiXmlElement**.

```
<bold>
```

Another **TiXmlElement**, this one a child of the "Item" element.

Etc.

Looking at the entire object tree, you end up with:

```
TiXmlDocument                                   "demo.xml"
        TiXmlDeclaration                        "version='1.0'" "standalone=no"
        TiXmlComment                            " Our to do list data"
        TiXmlElement                            "ToDo"
                TiXmlElement                    "Item" Attribtutes: priority = 1
                        TiXmlText                       "Go to the "
                        TiXmlElement            "bold"
                                TiXmlText               "Toy store!"
                TiXmlElement                    "Item" Attributes: priority=2
                        TiXmlText                       "Do bills"
```

Documentation

The documentation is build with Doxygen, using the 'dox' configuration file.

License

TinyXML is released under the zlib license:

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.

2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

3. This notice may not be removed or altered from any source distribution.

References

The World Wide Web Consortium is the definitive standard body for XML, and their web pages contain huge amounts of information.

The definitive spec: http://www.w3.org/TR/2004/REC-xml-20040204/

I also recommend "XML Pocket Reference" by Robert Eckstein and published by OReilly...the book that got the whole thing started.

Contributors, Contacts, and a Brief History

Thanks very much to everyone who sends suggestions, bugs, ideas, and encouragement. It all helps, and makes this project fun. A special thanks to the contributors on the web pages that keep it lively.

So many people have sent in bugs and ideas, that rather than list here we try to give credit due in the "changes.txt" file.

TinyXML was originally written by Lee Thomason. (Often the "I" still in the documentation.) Lee reviews changes and releases new versions, with the help of Yves Berquin, Andrew Ellerton, and the tinyXml community.

We appreciate your suggestions, and would love to know if you use TinyXML. Hopefully you will enjoy it and find it useful. Please post questions, comments, file bugs, or contact us at:

www.sourceforge.net/projects/tinyxml

Lee Thomason, Yves Berquin, Andrew Ellerton
