XML.APACHE.ORG WWW.APACHE.ORG WWW.W3.ORG

- Xerces Project
- Overview
- Charter
- Release Info
- Download

Xerces-C++ 3.1.2
- Installation
- Build Instructions
- Programming
- Samples
- FAQs
- API Reference
- DOM C++ Binding
- Migration Guide
- Feedback
- Bug-Reporting
- Mailing Lists
- Source Repository
- Applications

Questions

- Does Xerces-C++ support Schema?
- Why Xerces-C++ does not support this particular Schema feature?
- Why does my application crash when instantiating the parser?
- Is it OK to call the XMLPlatformUtils::Initialize/Terminate pair of routines multiple times in one program?
- Why does my application crash or hang if XMLPlatformUtils::Initialize()/Terminate() pair is called more than once?
- Why does my application crash after calling XMLPlatformUtils::Terminate()?
- I'm suddenly getting segfaults with Xerces-C 2.3.0; why might this be?
- Is Xerces-C++ thread-safe?
- I am seeing memory leaks in Xerces-C++. Are they real?
- I find memory leaks in Xerces-C++. How do I eliminate it?
- Can Xerces-C++ create an XML skeleton based on a DTD
- Can I use Xerces-C++ to perform write validation
- Can I validate the data contained in a DOM tree?
- How to write out a DOM tree into a string or an XML file?
- Why does DOMNode::cloneNode() not clone the pointer assigned to a DOMNode via DOMNode::setUserData()?
- How are entity reference nodes handled in DOM?
- What kinds of URLs are currently supported in Xerces-C++?
- How can I add support for URLs with HTTP/FTP protocols?
- Can I use Xerces-C++ to parse HTML?
- I keep getting an error: "invalid UTF-8 character". What's wrong?
- What encodings are supported by Xerces-C / XML4C?
- What character encoding should I use when creating XML documents?
- Is EBCDIC supported?
- Why does deleting a transcoded string result in assertion on windows?
- How do I transcode to/from something besides the local code page?

- Why does setProperty not work?
- Why does getProperty not work?
- Why does the parser still try to locate the DTD even validation is turned off and how to ignore external DTD reference?
- Why do I get segmentation fault when running on Redhat Linux?
- Why does the XML data generated by the DOMWriter does not match my original XML input?
- Why does my application crash when deleting the parser after releasing a document?
- Why do we have two versions of some XMLString methods (one with memory manager and one without)?

Does Xerces-C++ support Schema?

Yes. The Xerces-C++ 2.8.0 contains an implementation of the W3C XML Schema Language, a recommendation of the Worldwide Web Consortium available in three parts: XML Schema: Primer and XML Schema: Structures and XML Schema: Datatypes. We consider this implementation complete. See the Schema page for limitations.

Why Xerces-C++ does not support this particular Schema feature?

The Xerces-C++ 2.8.0 contains an implementation of the W3C XML Schema Language, a recommendation of the Worldwide Web Consortium available in three parts: XML Schema: Primer and XML Schema: Structures and XML Schema: Datatypes. We consider this implementation complete. See the Schema page for limitations.

If you find any Schema feature which is specified in the W3C XML Schema Language Recommendation does not work with Xerces-C++ 2.8.0, we encourage the submission of bugs as described in Bug-Reporting page.

Why does my application crash when instantiating the parser?

In order to work with the Xerces-C++ parser, you have to first initialize the XML subsystem. The most common mistake is to forget this initialization. Before you make any calls to Xerces-C++ APIs, you must call XMLPlatformUtils::Initialize():

```
try {
   XMLPlatformUtils::Initialize();
}
catch (const XMLException& toCatch) {
   // Do your failure processing here
}
```

This initializes the Xerces system and sets its internal variables. Note that you must the include `xercesc/util/PlatformUtils.hpp` file for this to work.

Is it OK to call the XMLPlatformUtils::Initialize/Terminate pair of routines multiple times in one program?

Yes. Since Xerces-C++ 1.5.2, the code has been enhanced so that calling XMLPlatformUtils::Initialize/Terminate pair of routines multiple times in one pr allowed.

But the application needs to guarantee that only one thread has entered eith XMLPlatformUtils::Initialize() or the method XMLPlatformUtils::Terminate() at

If you are calling XMLPlatformUtils::Initialize() a number of times, and then fo XMLPlatformUtils::Terminate() the same number of times, only the first XMLPlatformUtils::Initialize() will do the initialization, and only the last XMLPlatformUtils::Terminate() will clean up the memory. The other calls are i

To ensure all the memory held by the parser are freed, the number of XMLPlatformUtils::Terminate() calls should match the number of XMLPlatforr calls.

Consider the following code snippets (for illustration simplicity the following s coded in try/catch clause):

```
// The XMLPlatformUtils::Initialize/Terminate calls are paired.
{
    // Initialize the parser
    XMLPlatformUtils::Initialize();

    SAXParser* parser = new SAXParser;
    parser->parse(xmlFile);
    delete parser;

    // Free all memory that was being held by the parser
    XMLPlatformUtils::Terminate();

    // Initialize the parser
    XMLPlatformUtils::Initialize();

    parser = new SAXParser;
    parser->parse(xmlFile);
    delete parser;

    // Free all memory that was being held by the parser
    XMLPlatformUtils::Terminate();
}



// calls XMLPlatformUtils::Initialize() three times
// then calls XMLPlatformUtils::Terminate() numerous times
{
    // Initialize the parser
    XMLPlatformUtils::Initialize();

    // The next two calls are no-op
    XMLPlatformUtils::Initialize();
    XMLPlatformUtils::Initialize();

    SAXParser* parser = new SAXParser;
    parser->parse(xmlFile);
    delete parser;

    // The first two XMLPlatformUtils::Terminate() calls are no-op
    XMLPlatformUtils::Terminate();
    XMLPlatformUtils::Terminate();

    // This third XMLPlatformUtils::Terminate() will free all memory that was being
    XMLPlatformUtils::Terminate();

    // This extra fourth XMLPlatformUtils::Terminate() call is no-op.
    // However calling XMLPlatformUtils::Terminate() without a matching XMLPlatform
    // is dangerous and should be avoided.
    XMLPlatformUtils::Terminate();
}
```

Why does my application crash or hang if XMLPlatformUtils::Initialize()/Terminate() pair is called more than once?

Please make sure you are using the Xerces-C++ 1.5.2 or up.

Earlier version of Xerces-C++ does not allow XMLPlatformUtils::Initialize()/Terminate() pair to be called more than once or has a problem.

Why does my application crash after calling XMLPlatformUtils::Terminate()?

Please make sure the XMLPlatformUtils::Terminate() is the last Xerces-C++ function to be called in your program. NO explicit nor implicit Xerces-C++ destructor (those local data that are destructed when going out of scope) should be called after XMLPlatformUtils::Terminate().

For example consider the following code snippets which is incorrect (for illustration simplicity the following sample code is not coded in try/catch clause):

```
1: {
2:    XMLPlatformUtils::Initialize();
3:    DOMString c("hello");
4:    XMLPlatformUtils::Terminate();
5: }
```

The DOMString object "c" is destructed when going out of scope at line 5 before the closing brace. As a result, DOMString destructor is called at line 5 after XMLPlatformUtils::Terminate() which is wrong. Correct code should be:

```
1: {
2:    XMLPlatformUtils::Initialize();
2a:    {
3:           DOMString c("hello");
3a:    }
4:    XMLPlatformUtils::Terminate();
5: }
```

The extra pair of braces (line 2a and 3a) ensures that all implicit destructors are called before terminating Xerces-C++.

In addition the application also needs to guarantee that only one thread has entered either the method XMLPlatformUtils::Initialize() or the method XMLPlatformUtils::Terminate() at any one time.

I'm suddenly getting segfaults with Xerces-C 2.3.0; why might this be?

The introduction of pluggable memory management into Xerces-C, one of the main features of 2.3.0, means that application writers have to be more conscious about destructors being invoked implicitly after a call to XMLPlatformUtils::Terminate(). For example, the following code is guaranteed to produce a segmentation fault under Xerces-C 2.3.0, while it happened to work under previous versions (in fact, this was how our SAXPrint sample was formerly written; try-catch blocks removed for brevity):

```
void myParsingFunction()
{
    XMLPlatformUtils::Initialize();
    SAXParser parser;
    //parser.various method calls
    XMLPlatformUtils::Terminate();
} // seg fault here!
```

The reason this will produce a segmentation fault is that any dynamic memory the SAXParser (or any other of Xerces's parsers) needs to allocate is now allocated by default by a static object owned by XMLPlatformUtils. When the XMLPlatformUtils::Terminate() call is made, this object is destroyed--and, consequently, so are all the objects that it directly created. This includes all the objects dynamically allocated by the SAXParser. When the parser object goes out of scope, its destructor is invoked, and this attempts to destroy all the objects that it created--which have of course just been destroyed by the static MemoryManager in XMLPlatformUtils.

To avoid this, one must either explicitly scope the parser object inside calls to XMLPlatformUtils::Initialize() and XMLPlatformUtils::Terminate(), or dynamically allocate the parser object and destroy it explicitly before the call to XMLPlatformUtils::Terminate() is made.

Another way of producing segmentation faults--that again, unfortunately, was employed by some of our samples--is to have calls to XMLPlatformUtils::Terminate() in a catch block that catches any of Xerces's exceptions. Since the destructor of the exception will implicitly be invoked upon exit from the catch block, and since some of the exceptions' destructors call on Xerces's default memory manager to destroy dynamically-allocated objects, their destruction will provoke a segmentation fault even if a return statement is placed in the catch block since the default memory manager will no longer exist. This practice is now avoided in all our samples.

Is Xerces-C++ thread-safe?

This is not a question that has a simple yes/no answer. Here are the rules for using Xerces-C++ in a multi-threaded environment:

Within an address space, an instance of the parser may be used without restriction from a single thread, or an instance of the parser can be accessed from multiple threads, provided the application guarantees that only one thread has entered a method of the parser at any one time.

When two or more parser instances exist in a process, the instances can be used concurrently, without external synchronization. That is, in an application containing two parsers and two threads, one parser can be running within the first thread concurrently with the second parser running within the second thread.

The same rules apply to Xerces-C++ DOM documents. Multiple document instances may be concurrently accessed from different threads, but any given document instance can only be accessed by one thread at a time.

DOMStrings allow multiple concurrent readers. All DOMString const methods are thread safe, and can be concurrently entered by multiple threads. Non-const DOMString methods, such as `appendData()`, are not thread safe and the application must guarantee that no other methods (including const methods) are executed concurrently with them.

The application also needs to guarantee that only one thread has entered either the method XMLPlatformUtils::Initialize() or the method XMLPlatformUtils::Terminate() at any one time.

I am seeing memory leaks in Xerces-C++. Are they real?

The Xerces-C++ library allocates and caches some commonly reused items. The storage for these may be reported as memory leaks by some heap analysis tools; to avoid the problem, call the function `XMLPlatformUtils::Terminate()` before your application exits. This will free all memory that was being held by the library.

For most applications, the use of `Terminate()` is optional. The system will recover all memory when the application process shuts down. The exception to this is the use of Xerces-C++ from DLLs that will be repeatedly loaded and unloaded from within the same process. To avoid memory leaks with this kind of use, `Terminate()` must be called before unloading the Xerces-C++ library

To ensure all the memory held by the parser are freed, the number of XMLPlatformUtils::Terminate() calls should match the number of XMLPlatformUtils::Initialize() calls.

If you are using XML4C where ICU is used, you may call ICU function u_cleanup() to clean up ICU static data. Please see ICU documentation for details.

I find memory leaks in Xerces-C++. How do I eliminate it?

The "leaks" that are reported through a leak-detector or heap-analysis tools aren't really leaks in most application, in that the memory usage does not grow over time as the XML parser is used and re-used.

What you are seeing as leaks are actually lazily evaluated data allocated into static variables. This data gets released when the application ends. You can make a call to `XMLPlatformUtil::terminate()` to release all the lazily allocated variables before you exit your program.

To ensure all the memory held by the parser are freed, the number of XMLPlatformUtils::Terminate() calls should match the number of XMLPlatformUtils::Initialize() calls.

If you are using XML4C where ICU is used, you may call ICU function u_cleanup() to clean up ICU static data. Please see ICU documentation for details.

Is there a function that I have totally missed that creates an XML file from a DTD, (obviously with the values missing, a skeleton, as it were)?

No. This is not supported.

Can I use Xerces-C++ to perform "write validation" (which is having an appropriate Grammar and being able to add elements to the DOM whilst validating against the grammar)?

No. This is not supported.

The best you can do for now is to create the DOM document, write it back as XML and re-parse it.

Is there a facility in Xerces-C++ to validate the data contained in a DOM tree? That is, without saving and re-parsing the source document?

No. The best option for now is to generate XML source from the DOM and feed that back into the parser.

How to write out a DOM tree into a string or an XML file?

Please make sure you are using Xerces-C++ 2.8.0 or up.

You can use the DOMWriter::writeToString, or DOMWriter::writeNode to serialize a DOM tree. Please refer to the sample DOMPrint or the API documentation for more details of DOMWriter.

Why does DOMNode::cloneNode() not clone the pointer assigned to a DOMNode via DOMNode::setUserData()?

Xerces-C++ supports the DOMNode::userData specified in the DOM level 3 Node interface. As is made clear in the description of the behaviour of `cloneNode()`, userData that has been set on the Node is not cloned. Thus, if the userData is to be copied to the new Node, this copy must be effected manually. Note further that the operation of `importNode()` is specified similarly.

How are entity reference nodes handled in DOM?

If you are using the native DOM classes, the function `setCreateEntityReferenceNodes` controls how entities appear in the DOM tree. When setCreateEntityReferenceNodes is set to true (the default), an occurrence of an entity reference in the XML document will be represented by a subtree with an EntityReference node at the root whose children represent the entity expansion. Entity expansion will be a DOM tree representing the structure of the entity expansion, not a text node containing the entity expansion as text.

If setCreateEntityReferenceNodes is false, an entity reference in the XML document is represented by only the nodes that represent the entity expansion. The DOM tree will not contain any entityReference nodes.

What kinds of URLs are currently supported in Xerces-C++?

The `XMLURL` class provides for limited URL support. It understands the `file://`, `http://`, and `ftp://` URL types, and is capable or parsing them into their constituent components, and normalizing them. It also supports the commonly required action of conglomerating a base and relative URL into a single URL. In other words, it performs the limited set of functions required by an XML parser.

Another thing that URLs commonly do are to create an input stream that provides access to the entity referenced. The parser, as shipped, only supports this functionality on URLs in the form `file:///` and `file://localhost/`, i.e. only when the URL refers to a local file.

You may enable support for HTTP and FTP URLs by implementing and installing a NetAccessor object. When a NetAccessor object is installed, the URL class will use it to create input streams for the remote entities referred to by such URLs.

How can I add support for URLs with HTTP/FTP protocols?

Support for the http: protocol is now included by default on all platforms.

To address the need to make remote connections to resources specified using additional protocols, ftp for example, Xerces-C++ provides the `NetAccessor` interface. The header file is `src/xercesc/util/XMLNetAccessor.hpp`. This interface allows you to plug in your own implementation of URL networking code into the Xerces-C++ parser.

Can I use Xerces-C++ to parse HTML?

Yes, but only if the HTML follows the rules given in the XML specification. Most HTML, however, does not follow the XML rules, and will generate XML well-formedness errors.

I keep getting an error: "invalid UTF-8 character". What's wrong?

Most commonly, the XML `encoding =` declaration is either incorrect or missing. Without a declaration, XML defaults to the use utf-8 character encoding, which is not compatible with the default text file encoding on most systems.

The XML declaration should look something like this:

```
<?xml version="1.0" encoding="iso-8859-1"?>
```

Make sure to specify the encoding that is actually used by file. The encoding for "plain" text files depends both on the operating system and the locale (country and language) in use.

Another common source of problems is that some characters are not allowed in XML documents, according to the XML spec. Typical disallowed characters are control characters, even if you escape them using the Character Reference form. See the XML spec, sections 2.2 and 4.1 for details. If the parser is generating an `Invalid character (Unicode: 0x???)` error, it is very likely that there's a character in there that you can't see. You can generally use a UNIX command like "od -hc" to find it.

What encodings are supported by Xerces-C / XML4C?

Xerces-C has intrinsic support for ASCII, UTF-8, UTF-16 (Big/Small Endian), UCS4 (Big/Small Endian), EBCDIC code pages IBM037, IBM1047 and IBM1140 encodings, ISO-8859-1 (aka Latin1) and Windows-1252. This means that it can parse input XML files in these above mentioned encodings.

XML4C -- the version of Xerces-C available from IBM -- combines Xerces-C and International Components for Unicode (ICU) and extends the encoding support to over 100 different encodings that are allowed by ICU. In particular, all the encodings registered with the Internet Assigned Numbers Authority (IANA) are supported in XML4C.

Some implementations or ports of Xerces-C provide support for additional encodings. The exact set will depend on the supplier of the parser and on the character set transcoding services in use.

What character encoding should I use when creating XML documents?

The best choice in most cases is either utf-8 or utf-16. Advantages of these encodings include:

- The best portability. These encodings are more widely supported by XML processors than any others, meaning that your documents will have the best possible chance of being read correctly, no matter where they end up.
- Full international character support. Both utf-8 and utf-16 cover the full Unicode character set, which includes all of the characters from all major national, international and industry character sets.
- Efficient. utf-8 has the smaller storage requirements for documents that are primarily composed of characters from the Latin alphabet. utf-16 is more efficient for encoding Asian languages. But both encodings cover all languages without loss.

The only drawback of utf-8 or utf-16 is that they are not the native text file format for most systems, meaning that common text file editors and viewers can not be directly used.

A second choice of encoding would be any of the others listed in the table above. This works best when the xml encoding is the same as the default system encoding on the machine where the XML document is being prepared, because the document will then display correctly as a plain text file. For UNIX systems in countries speaking Western European languages, the encoding will usually be iso-8859-1.

The versions of Xerces distributed by IBM, both C and Java (known respectively as XML4C and XML4J), include all of the encodings listed in the above table, on all platforms.

A word of caution for Windows users: The default character set on Windows systems is windows-1252, not iso-8859-1. While Xerces-C++ does recognize this Windows encoding, it is a poor choice for portable XML data because it is not widely recognized by other XML processing tools. If you are using a Windows-based editing tool to generate XML, check which character set it generates, and make sure that the resulting XML specifies the correct name in the `encoding="..."` declaration.

Is EBCDIC supported?

Yes, Xerces-C++ supports EBCDIC with the ibm1140, ibm037 and ibm1047 encodings. When creating EBCDIC encoded XML data, the preferred encoding is ibm1140. The ibm037 encoding, and its alternate name, ebcdic-cp-us, is almost the same as ibm1140, but it lacks the Euro symbol.

These three encodings, ibm1140, ibm037 and ibm1047, are available on both Xerces-C and IBM XML4C, on all platforms.

On IBM System 390, XML4C also supports three alternative forms, ibm037-s390, ibm1140-s390, and ibm1047-s390. These are similar to the base ibm037, ibm1140, and ibm1047 encodings, but with alternate mappings of the EBCDIC new-line character, which allows them to appear as normal text files on System 390. These encodings are not supported on other platforms, and should not be used for portable data.

XML4C on System 390 and AS/400 also provides additional EBCDIC encodings, including those for the character sets of different countries. The exact set supported will be platform dependent, and these encodings are not recommended for portable XML data.

Why does deleting a transcoded string result in assertion on windows?

Both your application program and the Xerces-C++ DLL must use the same *DLL* version of the runtime library. If either statically links to the runtime library, the problem will still occur.

For example, for a Win32/VC6 build, the runtime library build setting MUST be "Multithreaded DLL" for release builds and "Debug

Multithreaded DLL" for debug builds.

Or for example for a Win32/BCB6 build, application need to switch to Multithreaded runtime to avoid such memory access violation.

To bypass such problem, instead of calling operator delete[] directly, you can use the provided function XMLString::release to delete any string that was allocated by the parser. This will ensure the string is allocated and deleted by the same DLL and such assertion problem should be resolved.

How do I transcode to/from something besides the local code page?

XMLString::transcode() will transcode from XMLCh to the local code page, ar which take a char* assume that the source text is in the local code page. If th must transcode the text yourself. You can do this using local transcoding sup such as Iconv on Unix or IBM's ICU package. However, if your transcoding n you can achieve some better portability by using the Xerces-C++ parser's tra You get a transcoder like this:

- Call XMLPlatformUtils::fgTransServer->MakeNewTranscoderFor() and of the encoding you wish to create a transcoder for. This will return a tra which you own and must delete when you are through with it. NOTE: Yo maximum block size that you will pass to the transcoder at one time, ar blocks of characters of this count or smaller when you do your transcoc this is that this is really an internal API and is used by the parser itself t The parser always does transcoding in known block sizes, and this allo be much more efficient for internal use since it knows the max size it wi with and can set itself up for that internally. In general, you should stick the 4 to 64K range.
- The returned transcoder is something derived from XMLTranscoder, so returned to you via that interface.
- This object is really just a wrapper around the underlying transcoding sy use by your version of Xerces, and does whatever is necessary to hanc between the XMLCh representation and the representation used by tha transcoding system.
- The transcoder object has two primary APIs, transcodeFrom() and tran transcode between the XMLCh format and the encoding you indicated.
- These APIs will transcode as much of the source data as will fit into the you provide. They will tell you how much of the source they ate and hov target they filled. You can use this information to continue the process u consumed.
- char* data is always dealt with in terms of bytes, and XMLCh data is alv terms of characters. Don't mix up which you are dealing with or you will results, since many encodings don't have a one to one relationship of c
- When transcoding from XMLCh to the target encoding, the transcodeTo an 'unrepresentable flag' parameter, which tells the transcoder how to c XMLCh code point that cannot be converted legally to the target encod

easily happen since XMLCh is Unicode and can represent thousands c
options are to use a default replacement character (which the underlyir
service will choose, and which is guaranteed to be legal for the target e
throw an exception.

Here is an example:

```
// create an XMLTranscoder that is able to transcode between Unicode and Big5
// ASSUMPTION: assumes your underlying transcoding utility supports this encoding B
XMLTranscoder* t =
    XMLPlatformUtils::fgTransService->makeNewTranscoderFor("Big5", failReason, 16*1

// source string is in Unicode, wanna to transcode to Big5
t->transcodeTo(source_unicode, length, result_Big5, length, charsEaten, XMLTranscod

// source string in Big5, wanna to transcode to Unicode
t->transcodeFrom(source_Big5, length, result_unicode, length, bytesEaten, (unsigned
```

Why does setProperty not work?

The function `SAX2XMLReader::setProperty(const XMLCh* const name, void* value)` and `DOMBuilder::setProperty(const XMLCh* const name, void* value)` takes a void pointer for the property value. Application is required to initialize this void pointer to a correct type. See SAX2 Programming Guide and DOM Programming Guide to learn exactly what type of property value that each property expects for processing. Passing a void pointer that was initialized with a wrong type will lead to unexpected result.

Why does getProperty not work?

The function `void* SAX2XMLReader::getProperty(const XMLCh* const name)` and `void* DOMBuilder::getProperty(const XMLCh* const name)` returns a void pointer for the property value. See SAX2 Programming Guide and exactly what type of object each property returns.

The parser owns the returned pointer. The memory allocated for the returned pointer will be destroyed when the parser is deleted. To ensure accessibility of the returned information after the parser is deleted, callers need to copy and store the returned information somewhere else; otherwise you may get unexpected result. Since the returned pointer is a generic void pointer, see SAX2 Programming Guide and DOM Programming Guide to learn exactly what type of property value each property returns for replication.

Why does the parser still try to locate the DTD even validation is turned off and how to ignore external DTD

reference?

When DTD is referenced, the parser will try to read it, because DTDs can provide a lot more information than just validation. It defines entities and notations, external unparsed entities, default attributes, character entities, etc... So it will always try to read it if present, even if validation is turned off.

To ignore the DTD, with Xerces-C++ 2.8.0 or up, you can call `setLoadExternalDTD(false)` (or `setFeature(XMLUni::fgXercesLoadExternalDTD, false)` to disable the loading of external DTD. The parser will then ignore any external DTD completely if the validationScheme is set to Val_Never.

Note: This flag is ignored if the validationScheme is set to Val_Always or Val_Auto.

To ignore the DTD in earlier version of Xerces-C++, the only way to get around this is to install an EntityResolver (see the Redirect sample for an example of how this is done), and reset the DTD file to "".

Why do I get segmentation fault when running on Redhat Linux?

There were some problems with Redhat Linux 7.x with C++ exception handling across shared libraries. More details can be found here. Please try to upgrade your Redhat Linux gcc to the latest patch level and see if it helps.

Why does the XML data generated by the DOMWriter does not match my original XML input?

If you parse an xml document using XercesDOMParser or DOMBuilder and pass such DOMNode to DOMWriter for serialization, you may not get something that is exactly the same as the original XML data. The parser may have done normalization, end of line conversion, or has expanded the entity reference as per the XML 1.0 spec, 4.4 XML Processor Treatment of Entities and References. From DOMWriter perspective, it does not know what the original string was, all it sees is a processed DOMNode generated by the parser. But since the DOMWriter is supposed to generate something that is parsable if sent back to the parser, it will not print the DOMNode node value as is. The DOMWriter may do some "touch up" to the output data for it to be parsable.

See How does DOMWriter handle built-in entity Reference in node value? to understand further how DOMWriter touches up the entity reference.

Why does my application crash when deleting the parser after releasing a document?

In most cases, the parser handles deleting documents when the parser gets deleted. However, if an application needs to release a document, it shall adopt the document before releasing it, so that the parser knows that the ownership of this particular document is transfered to the application and will not try to delete it once the parser gets deleted.

```
XercesDOMParser *parser = new XercesDOMParser;
...
try
{
    parser->parse(gXmlFile);
}
catch ()
{
...
}
DOMNode *doc = parser->getDocument();
...
parser->adoptDocument();
doc->release();
...
delete parser;
```

The alternative to release document is to call parser's resetDocumentPool(), which releases all the documents parsed.

Why do we have two versions of some XMLString methods (one with memory manager and one without)?

With the introduction of the configurable memory manager, we didn't want to break users by changing the signature of the existing methods (for example, transcode and replicate). Also, we did not want to provide a default memory manager as it would introduce a side effect with users experiencing some strange core dumps. The latter will occur when the scope of the string allocated is beyond that of XMLPlatformUtils::Terminate (i.e. a string is allocated using the default memory manager which is deleted when XMLPlatformUtils::Terminate is called, but the allocated string is deleted later). We plan to deprecate the methods without a memory manager in a later release.