Welcome to XML::Xerces

Table of contents

1. Xerces Perl: The Perl API to the Apache Xerces XML parser

1.1. Current Release: XML::Xerces 2.7.0-0

XML::Xerces is the Perl API to the Apache project's Xerces XML parser. It is implemented using the Xerces C++ API, and it provides access to *most* of the C++ API from Perl.

Because it is based on Xerces-C, XML::Xerces provides a validating XML parser that makes it easy to give your application the ability to read and write XML data. Classes are provided for parsing, generating, manipulating, and validating XML documents. XML::Xerces is faithful to the XML 1.0 recommendation and associated standards (DOM levels 1, 2, and 3, SAX 1 and 2, Namespaces, and W3C XML Schema). The parser provides high performance, modularity, and scalability, and provides full support for Unicode.

XML::Xerces implements the vast majority of the Xerces-C API (if you notice any discrepancies please mail the list). The exception is some functions in the C++ API which either have better Perl counterparts (such as file I/O) or which manipulate internal C++ information that has no role in the Perl module.

The majority of the API is created automatically using Simplified Wrapper Interface Generator (SWIG). However, care has been taken to make most method invocations natural to perl programmers, so a number of rough C++ edges have been smoothed over (See the Special Perl API Features section).

1.2. Support

The online users mailing list is the place for any questions. It is at: p-dev@xerces.apache.org

1.3. Available Platforms

The code has been tested on the following platforms:

- Linux
- Cygwin
- Windows
- Mac OS X
- BSD
- Solaris
- AIX
- Tru64

1.4. Build Requirements

1.4.1. ANSI C++ compiler

Builds are known to work with the GNU C compiler, and other platform specific compilers (such as VC++ on Windows and Forte on Solaris). Contributions in this area are always welcome :-).

1.4.2. Perl5

Note:
Required version: 5.6.0

XML::Xerces now supports Unicode. Since Unicode support wasn't added to Perl until 5.6.0, you will need to upgrade in order to use this and future versions of XML::Xerces. Upgrading to at least to the latest stable release, 5.6.1, is recommended.

If you plan on using Unicode, I *strongly* recommend upgrading to Perl-5.8.x, the latest stable version. There have been significant improvements to Perl's Unicode support.

1.4.3. The Apache Xerces C++ XML Parser

Note:
Required version: 2.7.0

(which can be downloaded from the apache archive) You'll need both the library and header files, and to set up any environment variables that will direct the XML::Xerces build to the directories where these reside.

1.5. Prepare for the build

1.5.1. Download XML::Xerces

Download the release and it's digital signature, from the apache Xerces-C archive.

1.5.2. Verify the archive

Optionally verify the release using the supplied digital signature (see the apache Xerces-Perl archive for details)

1.5.3. Unpack the archive

Unpack the archive in a directory of your choice. Example (for UNIX):

- `tar zxvf XML-Xerces-2.7.0-0.tar.gz`
- `cd XML-Xerces-2.7.0-0`

1.5.4. Getting Xerces-C

If the Xerces-C library and header files are installed on your system directly, e.g. via an rpm or deb package, proceed to the directions for building XML::Xerces.

Otherwise, you must download Xerces-C from www.apache.org. If there is a binary available for your architecture, you may use it, otherwise you must build it from source. If you wish to make Xerces-C available to other applications, you may install it however it is not necessary to do so in order to build XML::Xerces. To build XML::Xerces from an uninstalled Xerces-C set the XERCESCROOT environment variable the top-level directory of the source directory (i.e. the same value it needs to be to build Xerces-C):

```
export XERCESCROOT=/home/jasons/xerces-2.7.0/
```

OPTIONAL: If you choose to install Xerces-C on your system, you need to set the XERCES_INCLUDE and XERCES_LIB environment variables:

```
export XERCES_INCLUDE=/usr/include/xerces
export XERCES_LIB=/usr/lib
```

1.6. Build XML::Xerces

1. Go to the XML-Xerces-2.7.0-0 directory.
2. Build XML::Xerces as you would any perl package that you might get from CPAN:
3. - `perl Makefile.PL`
   - `make`
   - `make test`
   - `make install`

1.7. Using XML::Xerces

XML::Xerces implements the vast majority of the Xerces-C API (if you notice any discrepancies please mail the list). Documentation for this API are sadly not available in POD format, but the Xerces-C html documentation is available online.

For more information, see the examples in the samples/ directory. and the test scripts located in the t/ directory.

1.8. Special Perl API Features

Even though XML::Xerces is based on the C++ API, it has been modified in a few ways to make it more accessible to typical Perl usage, primarily in the handling:

- String I/O (Perl strings versus XMLch arrays)
- List I/O (Perl lists versus DOM_NodeList's)
- Hash I/O (Perl hashes versus DOM_NamedNodeMap's)
- Combined List/Hash classes
- void* handling
- DOM Serialization API
- Implementing Perl handlers for C++ event callbacks
- handling C++ exceptions
- XML::Xerces::XMLUni unicode constants

1.8.1. String I/O

The native data type for Xerces-C is the XMLCh* which is a UTF16 encoded string and in Perl strings are encoded in UTF8. All conversion back and forth between Perl and Xerces-C is handled automatically by XML::Xerces.

In fact a lot of effort is made to convert Perl variables into strings before passing them to Xerces-C. So any method that accepts an XMLCh* in Xerces-C will accept any non-undef value using Perl's built-in stringification mechanism.

1.8.2. List I/O

Any function that in the C++ API returns a `DOMNodeList` (e.g. `getChildNodes()` and `getElementsByTagName()` for example) will return different types if they are called in a list context or a scalar context. In a scalar context, these functions return a reference to a `XML::Xerces::DOMNodeList`, just like in C++ API. However, in a list context they will return a Perl list of `XML::Xerces::DOM_Node` references. For example:

```
# returns a reference to a XML::Xerces::DOMNodeList
my $node_list_ref = $doc->getElementsByTagName('foo');

# returns a list of XML::Xerces::DOMNode's
my @node_list = $doc->getElementsByTagName('foo');
```

1.8.3. Hash I/O

Any function that in the C++ API returns a `DOMNamedNodeMap` (`getEntities()` and `getAttributes()` for example) will return different types if they are called in a list context or a scalar context. In a scalar context, these functions return a reference to a `XML::Xerces::DOMNamedNodeMap`, just like in C++ API. However, in a list context they will return a Perl hash. For example:

```
            # returns a reference to a XML::Xerces::DOMNamedNodeMap
            my $attr_map_ref = $element_node->getAttributes();

            # returns a hash of the attributes
            my %attrs = $element_node->getAttributes();
```

1.8.4. Combined List/Hash classes (XMLAttDefList)

Any function that in the C++ API returns a XMLAttDefList (getAttDefList() for SchemaElementDecl and DTDElementDecl), will always return an instance of XML::Xerces::XMLAttDefList. However, there are two Perl specific API methods that can be invoked on the object: to_list() and to_hash().

```
            # get the XML::Xerces::XMLAttDefList.
      my $attr_list = $element_decl->getAttDefList();

      # return a list of XML::Xerces::XMLAttDef instances
            my @list = $attr_list->to_list();

            # returns a hash of the attributes, where the keys are the
            # result of calling getFullName() on the attributes, and the
            # values are the XML::Xerces::XMLAttDef instances.
            my %attrs = $attr_list->to_hash();
```

1.8.5. Void* handling

Any function in the C++ API that accepts a void*, for example setProperty() in DOMBuilder and SAX2XMLReader, must be handled specially. Currently, all void* methods convert their arguments to a string before passing them to Xerces-C. In the future, when other data types are needed, this functionality will be expanded. If you locate a case in which you need this support, please alert the development team (p-dev@xerces.apache.org).

1.8.6. Serialize API

The DOMWriter class is used for serializing DOM hierarchies. See t/DOMWriter.t or

samples/DOMPrint.pl for details.

For less complex usage, just use the serialize() method defined for all DOMNode subclasses.

1.8.7. Implementing {Document,Content,Error}Handlers from Perl

Thanks to suggestions from Duncan Cameron, XML::Xerces now has a handler API that matches the currently used semantics of other Perl XML API's. There are three classes available for application writers:

- PerlErrorHandler (SAX 1/2 and DOM 1)
- PerlDocumentHandler (SAX 1)
- PerlContentHandler (SAX 2)

Using these classes is as simple as creating a perl subclass of the needed class, and redefining any needed methods. For example, to override the default fatal_error() method of the PerlErrorHandler class we can include this piece of code within our application:

```
package MyErrorHandler;
@ISA = qw(XML::Xerces::PerlErrorHandler);
sub fatal_error {die "Oops, I got an error\n";}

package main;
my $dom = new XML::Xerces::DOMParser;
$dom->setErrorHandler(MyErrorHandler->new());
```

1.8.8. Handling exceptions ({XML,DOM,SAX}Exception's)

Some errors occur outside parsing and are not caught by the parser's ErrorHandler. XML::Xerces provides a way for catching these errors using the PerlExceptionHandler class. Usually the following code is enough for catching exceptions:

```
eval{$parser->parser($my_file)};
XML::Xerces::error($@) if $@;
```

Wrap any code that might throw an exception inside an eval{...} and call XML::Xerces::error() passing $@, if $@ is set.

There are a default methods that prints out an error message and calls die(), but if more is needed, see the files t/XMLException.t, t/SAXException.t, and t/DOMException.t for details on how to roll your own handler.

1.8.9. XML::Xerces::XMLUni unicode constants

XML::Xerces uses many constant values for setting of features, and properties, such as for XML::Xerces::SAX2XMLReader::setFeature(). You can hard code the strings or integers into your programs but this will make them vulnerable to an API change. Instead, use the constants defined in the XML::Xerces::XMLUni class. If the API changes, the constants will be updated to reflect that change. See the file docs/UMLUni.txt for a complete listing of the constant names and their values.

1.9. Sample Code

XML::Xerces comes with a number of sample applications:

- SAXCount.pl: Uses the SAX interface to output a count of the number of elements in an XML document
- SAX2Count.pl: Uses the SAX2 interface to output a count of the number of elements in an XML document
- DOMCount.pl: Uses the DOM interface to output a count of the number of elements in an XML document
- DOMPrint.pl: Uses the DOM interface to output a pretty-printed version of an XML file to STDOUT
- DOMCreate.pl: Creates a simple XML document using the DOM interface and writes it to STDOUT
- DOM2hash.pl: Uses the DOM interface to convert the file to a simple hash of lists representation
- EnumVal.pl: Parses and input XML document and outputs the DTD information to STDOUT
- SEnumVal.pl: Parses and input XML document and outputs the XML Schema information to STDOUT

1.10. Development Tools

Note:

These are only for internal XML::Xerces development. If your intention is solely to use XML::Xerces to write XML applications in Perl, you will *NOT* need these tools.

1.10.1. SWIG

Simplified Wrapper Interface Generator (SWIG) is an open source tool by David Beazley of the University of Chicago for automatically generating Perl wrappers for C and C++ libraries (i.e. *.a or *.so for UNIX, *.dll for Windoes). You can get the source from the SWIG home page and then build it for your platform.

You will only need this if the include Xerces.C and XML::Xerces files do not work for your perl distribution. The pre-generated files have been created by SWIG 1.3 and work under Perl-5.6 or later.

This port will only work with SWIG 1.3.28 or later.

If your planning to use SWIG, you can set the environment variable SWIG to the full path to the SWIG executable before running `perl Makefile.pl`. For example:

```
export SWIG=/usr/local/bin/swig
```

This is only necessary if it isn't in your path or you have more than one version installed.

1.11. Acknowledgements

The Xerces development team would like to provide special acknowledgment to the following companies for their gracious financial support:

BBC: XML-Xerces-2.7

The British Broadcasting Corporation provided support for the XML-Xerces-2.7 release that has enabled patching a number of SAX-related Unicode bugs.

Cluster Technology: XML-Xerces-2.6

Cluster Technology Limited provided support for the XML-Xerces-2.6 release and for invaluable testing to help eliminate the major memory leaks that existed prior to the 2.6 release.