

XML parser API library for Netopeer

Petr Novák

15. 11. 2003

1 Motivation

The Netopeer router configuration system has several front-ends that use the same operations on a XML document. This API library provides a parser independent (this is very useful, because XML is a new technology and parsers have not stable API, some their functions do not conform to W3C standards or their behaviour is slightly different from what we need) and implementation/reimplementation of some methods we need (e. g., other devide elements, id generating, validation etc.).

This library wraps the current XML parser (Libxml2 at this time) and provides all the methods we need for front-ends. Front-ends can use a direct access to the Libxml2 library or work with the XML document using another parser library, but it is not recommended, because the parser independence is then lost.

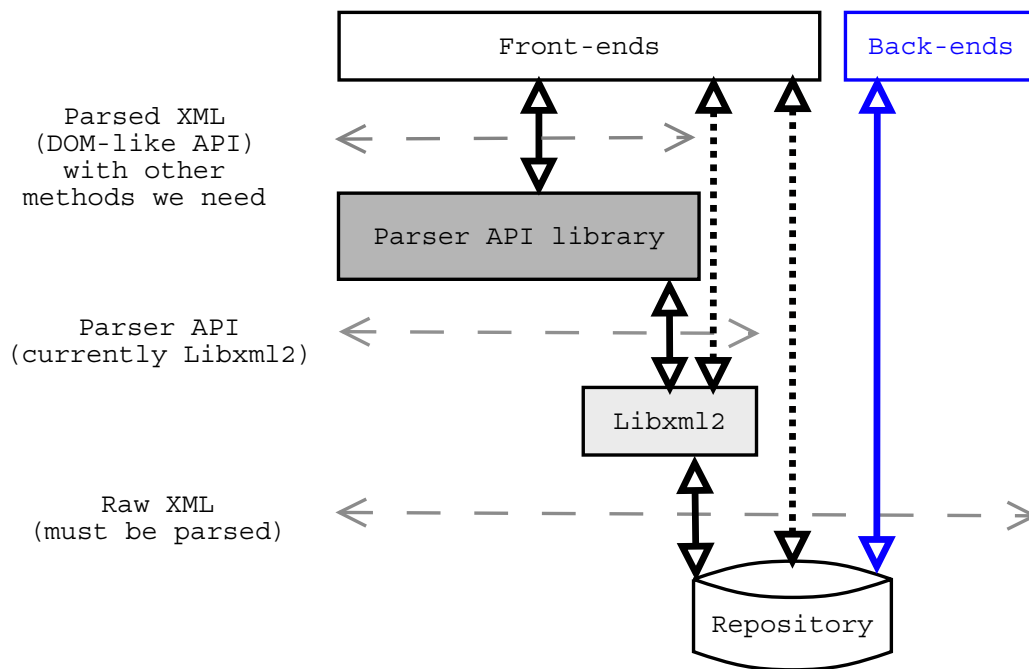


Figure 1: Component hierarchy

2 Description

API library provides the following methods for working with the XML document:

- Basic operations on XML files (open, create, modify and write).
- On-demand validation (e. g., for verification before writing or after adding some new nodes).
- Finding elements and attributes in the document using the XPath.
- DTD and Relax NG support.

XML tags are divided into these groups:

- Document
- Element
- Attribute

3 Detailed description

3.1 Schema support

API supports DTD and Relax NG. Relax NG is used implicitly, if DTD is used, a DTD directive must to be defined:

```
#define DTD
```

Note: The current Netopeer schema is defined using DTD, so the DTD directive is defined in the API library header file (*parser.h*). It will be removed in the future.

3.2 Data types and constants

TExceptionSubType: Contains all kinds of exceptions. Available values are:
NO_ERROR
API_EXCEPTION For bad parameters of API library methods call.
PARSER_ERROR For some errors in the parser library (initialization errors etc.) - Libxml2 at this time.
UNDEFINED_ERROR

TExceptionType: Type of an exception. All types are contained in the header file *parser-exception.h*.

TException: Class representing exception that was raised in the parser API library and programs using this library.

TDocument: Class representing whole XML document.

TElement: Class representing the XML element.

TAttribute: Class representing the XML attribute.

TElementsList: Vector of the items of TElement.

TAttributesList: Vector of the items of TAttribute.

3.3 Classes overview

3.3.1 TException

Purpose: Error reporting.

Description: Provides exceptions (error messages) from the API library (e. g., an incorrect value in some method), current parser (Libxml2 at this time) and unidentified errors. Each exception has its own exception subtype (what kind of an exception it is), type (error number) and a text message.

All exception types and subtypes are listed in the header file *parser-exceptions.h*.

Methods:

TException(const string text Message, const TExceptionSubType subType, const TExceptionType type):

Constructor. This method is used for raising an exception in the API library.

string getMessage(void): Returns the text exception (error) message.

TExceptionSubType getSubType(void): Returns the exception subtype (what kind of exception it is).

TExceptionType getType(void): Returns the exception type (error number).

3.3.2 TDocument

Purpose: Operations on the whole XML document.

Description: Includes methods for working with the whole document, such as creating a new document, open an existing document, writing a document to the file and validating it.

It supports an automatic id attribute generation for each created element. The name of the id attribute can be changed by setting the ID const in the *parser.h* header file (implicit name is 'id'). The value is an integer number.

Note: The sequence of id attribute values need not always be increasing. If any elements are removed, their id values are marked as empty in the next parsing of the document and are used at creating the new elements.

Methods:

TDocument(const string DocumentName, const string GlobalGrammarPath, const string LocalGrammarPath):

Constructor for creating the new document.

Note: You must enter only one of the schema paths.

TDocument(const string XMLFilePath, const string GrammarPath):

Constructor for parsing an existing document. If the document is using DTD as its schema, the second parameter cannot be set (DTD must be defined in the XML document).

~TDocument(void): Destructor.

Note: If this destructor is used, the entire document is erased and all elements linking to some part of this document became unreachable.

void write(const string XMLFileName): Writes the document to the file.

If the XMLFileName is entered, the document is saved to the file with this name, otherwise the file name is the internal document name (If the document was parsed from file, its file name becomes the internal document name. If the document was created as a new, the internal name of the document is a root element name with *.xml* suffix.).

Note: Validation is not executed in this method. You must call it yourself before writing.

bool validate(TElement &badElement, TAttribute &badAttribute):

Returns true, if the document is valid according to the schema (DTD or Relax NG). If the document is not valid, first non-valid element or first non-valid attribute is returned as a parameter. Always only one of them is returned, the second is empty (you can use *exist* method to check it). If the document is valid, both parameters are empty. When the document

is not valid and the error is somewhere else than in some element or attribute, both parameters are empty too.

TElement getRootElement(void): Returns the root element of the document.

3.3.3 TElement

Purpose: Provides operations on a single XML element.

Description: Includes elementary operation for reading element information, editing it, editing element attributes and searching elements.

Note: The term element in this context really means an XML element (i. e., not node with an attribute, CDATA, processing instruction or other type).

Methods:

TElement(void): Implicit constructor.

void blank(void): Cleans current TElement instance and sets it to an empty.

Note: It cleans just a link from the current TElement instance. If you can erase the real element from the XML document, use `erase`.

Before using a cleared TElement instance, you must assign it a new value first.

bool exist(void): Returns true if the current TElement instance is not empty.

string getName(void): Returns a working element name.

string getValue(void): Returns the character data content of the working element. Subelements and their contents are not included. In the case of a mixed content, the text nodes of the working element, which are interspersed by subelements, are concatenated and returned. All whitespace are preserved.

For example, consider the following part of the XML document:

```
<element1>
    value_part1
    <element2>
        another_value
    </element2>
    value_part2
</element1>
```

If we call `getValue` for the element `element1`, we obtain:

value_part1

value_part2

void setValue(const string Value): Sets the character data value of the working element.

Note: It removes all existing text nodes in this element and creates a single new text node with the given value.

bool hasChild(void): Returns true if the working element has any child elements.

Note: Child element in this context means only an XML element, not attribute node, text node, processing instruction etc.

TElementsList getChilds(const string ChildName): Returns child elements of the working element. If ChildName is specified, only elements with this name are returned.

Note: Child element in this context means only an XML element, not attribute node, text node, processing instruction etc.

TElement parentElement(void): Returns a parent element of the working element.

If this method is called on the root element, it returns an empty element.

void erase(const TElement): Erases the selected child element of the working element. All its childs are erased too.

TElement create(const string ElementName, const bool CreateId):
Creates a new child element of the working element and returns it. If CreateId is true, id attribute with the unique id for this new element is created.

TElement createBefore(const TElementsList ElementsBefore, const string ElementName, const bool CreateId):

Creates a new child element of the working element and returns it. If CreateId is true, id attribute with the unique id for this new element is created.

This created element is inserted before all elements in the ElementsBefore. If the ElementsBefore list is empty, the created element is inserted as the first child element of the working element.

TElementsList find(const string XPath): Finds items through the XPath expression in the XML tree.

The search context starts at the the current element. Use an absolute path in the XPath expression if you want to search the entire document.

TAttribute getAttribute(const string AttributeName): Gets an selected attribute of the working element.

TAttributesList getAttributes(void): Gets a list of all attributes of the working element.

TAttribute createAttribute(const string AttributeName, const string AttributeValue):

Creates a new working element attribute with the given name and value.

void eraseAttribute(const TAttribute ErasedAttribute): Erases the selected attribute of the working element.

string getAttributeValue(const string AttributeName): Returns the value of the selected attribute of the working element.

void setAttributeValue(const string AttributeName, const string AttributeValue):

Sets the value of the selected attribute in the working elemnet.

If the attribute does not exist, it is created.

3.3.4 TAttribute

Purpose: Provides an operation for attribute management.

Description: Includes an elementary operations for reading attribute information and editing it.

Methods:

TAttribute(void): Implicit constructor.

void blank (void): Cleans current TAttribute instance and sets it to empty.

Note: It cleans just a link from the current TAttribute instance. If you want to erase a real attribute from am XML document, use `eraseAttribute` on the parent element

Before using a cleared TAttribute instance, you must assign it a new value first

bool exist (void): Returns true if the current TAttribute instance is not empty.

string getName(void): Returns the attribute name.

string getValue(void): Returns the attribute value.

`void setValue(const string Value):` Sets the value of the attribute. Whitespace in the value is normalized automatically.

4 Installation

4.1 Requirements

Platform: C++ (tested on g++ 3.2.0)

Parser: Libxml2 version 2.5.0 and greater (tested on Libxml2-2.5.11)

4.2 Description of the files

parser.cpp - API library source code

parser.h - API library header file

parser-exception.h - API library exception list

Makefile

4.3 Preparation for the installation

You must set the LIBXML2 variable in the Makefile to the path where you have installed Libxml2.

4.4 Installation

Now you can compile the API library using:

```
$ make
```

This will create the shared library *libparser.so.0.0.1* in the current directory and the symbolic link to it named *libparser.so.0*. You can erase an temporary object file using:

```
$ make clean
```

5 Using

5.1 Changing the behaviour

You can change the schema and the name of the id attribute that is generated for a new elements. Both are specified and can be changed in the header file *parser.h*. It must be done before the compilation.

5.2 Your source code

You must import the API library header file using

```
#include "path/parser.h"
```

or

```
#include <path/parser.h>
```

if you have it in the standard header files directory.

Note: Some methods can raise an exception if they are called with bad parameters or Libxml2 is not correctly compiled. You must catch it or your program will be terminated.

5.3 Your source code compilation

This library can be compiled using the usual procedure, you must only set the path to the header files and libraries of the Libxml2, if you do not have it in the standard directories.

6 Conclusion

The actual syntax is documented in the header file *parser.h*.