NanoXML/Java 2.1

Marc De Scheemaeker
Copyright ©2001 Marc De Scheemaeker. All Rights Reserved

This document is provided “as-is”, without any express or implied warranty.
In no event will the author be held liable for any damages arising from the use of
this document or the software it describes.

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

1. The origin of this document must not be misrepresented; you must not claim
   you wrote the original document.

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

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

CHAPTER 1  Introduction  1
   About XML  1
   About NanoXML  2
   NanoXML 2  3
   NanoXML Extension to the XML System ID  3

CHAPTER 2  Retrieving Data From An XML Datasource  5
   A Very Simple Example  5
   Analyzing The Data  6
   Generating XML  8
   Namespaces  9

CHAPTER 3  Retrieving Data From An XML Stream  13
   The XML Builder  13
# Contents

Registering an XML Builder  15

## CHAPTER 4

**Advanced Topics**  17

- The NanoXML Reader  19
- The NanoXML Parser  20
- The NanoXML Validator  20
- The NanoXML Entity Resolvers  21
- The NanoXML Builder  23

## CHAPTER 5

**Class Reference**  31

- net.n3.nanoxml.IXMLBuilder (interface)  31
- net.n3.nanoxml.IXMLElement (interface)  36
- net.n3.nanoxml.IXMLEntityResolver (interface)  46
- net.n3.nanoxml.IXMLParser (interface)  48
- net.n3.nanoxml.IXMLReader (interface)  50
  - 50
- net.n3.nanoxml.IXMLValidator (interface)  53
- net.n3.nanoxml.NonValidator  56
- net.n3.nanoxml.StdXMLBuilder  59
- net.n3.nanoxml.StdXMLParser  64
- net.n3.nanoxml.StdXMLReader  66
- net.n3.nanoxml.ValidatorPlugin  71
- net.n3.nanoxml.XMLElement  78
- net.n3.nanoxml.XMLEntityResolver  89
- net.n3.nanoxml.XMLException  91
- net.n3.nanoxml.XMLParseException  94
- net.n3.nanoxml.XMLParserFactory  95
- net.n3.nanoxml.XMLValidationException  96
- net.n3.nanoxml.XMLWriter  98

---

NanoXML/Java 2.1
CHAPTER 1

Introduction

This chapter gives a short introduction to XML and NanoXML.

About XML

The extensible markup language, XML, is a way to mark up text in a structured document.

XML is a simplification of the complex SGML standard. SGML, the Standard Generalized Markup Language, is an international (ISO) standard for marking up text and graphics. The best known application of SGML is HTML.

Although SGML data is very easy to write, it's very difficult to write a generic SGML parser. When designing XML however, the authors removed much of the flexibility of SGML making it much easier to parse XML documents correctly.

XML data is structured as a tree of entities. An entity can be a string of character data or an element which can contain other entities. Elements can optionally have a set of attributes. Attributes are key/value pairs which set some properties of an element. The following example shows some XML data:
Introduction

<book>
  <chapter id="my chapter">
    <title>The title</title>
    Some text.
  </chapter>
</book>

At the root of the tree, you can find the element "book". This element contains one child element: "chapter". The chapter element has one attribute which maps the key "id" to "my chapter". The chapter element has two child entities: the element "title" and the character data "Some text.". Finally, the title element has one child, the string "The title".

About NanoXML

In April 2000, NanoXML was first released as a spin-off project of AUIT, the Abstract User Interface Toolkit.

The intent of NanoXML was to be a small parser which was easy to use. SAX and DOM are much too complex for what I needed and the mainstream parsers were either much too big or had a very restrictive license.

NanoXML 1 has all the features I needed: it is very small (about 6K), is reasonably fast for small XML documents, is very easy to use and is free (zlib/libpng license). As I never intended to use NanoXML to parse DocBook documents, there was no support for mixed data or DTD parsing.

NanoXML was released as a SourceForge project and, because of the very good response from its users, it matured to a small and stable parser. The final version, release 1.6.8 was released in May 2001.

Because of its small size, people started to use NanoXML for embedded systems (KVM, J2ME) and kindly submitted patches to make NanoXML work in such restricted environment.
NanoXML 2

In July 2001, NanoXML 2 has been released. Unlike NanoXML 1, speed and XML compliancy were considered to be very important when the new parser was designed. NanoXML 2 is also very modular: you can easily replace the different components in the parser to customize it to your needs. The modularity of NanoXML 2 also benefits extensions like e.g. SAX support which can now directly access the parser. In NanoXML 1, the SAX adapter had to iterate the data structure built by the base product.

Although many features were added to NanoXML, the second release was still very small. The full parser with builder fits in a JAR file of about 32K. This is still very tiny, especially when you compare this with the "standard" parsers of more than four times its size.

As there is still need for a tiny parser like NanoXML 1, there is a special branch of NanoXML 2: NanoXML/Lite. This parser is source compatible with NanoXML 1 but features a new parsing algorithm which makes it more than twice as fast as the older version. It is however more restrictive on the XML data it parses: the older version allowed some not-wellformed data to be parsed.

There are three branches of NanoXML 2:

- NanoXML/Lite is the successor of NanoXML 1. It features an almost compatible parser which is extremely small.
- NanoXML/Java is the standard parser.
- NanoXML/SAX is the SAX adapter for NanoXML/Java.

The latest version of NanoXML is NanoXML 2.1.1, which is released in November 2001.

NanoXML Extension to the XML System ID

Because it's convenient to put data files into jar files, we need some way to specify that we want some resource which can be found in the class path. There is no support for such resources in the XML 1.0 specification. NanoXML allows you to specify such resources using the reference part of a URL.
This means that if the DTD of the XML data is put in the resource /data/foo.dtd, you can specify such path using the following document type declaration:

```xml
<!DOCTYPE foo SYSTEM 'file:///data/foo.dtd'>
```

It’s even possible to specify a resource found in a particular jar, like in the following example:

```xml
<!DOCTYPE foo SYSTEM 'http://myserver.com/dtds.jar#/foo.dtd'>
```
CHAPTER 2

Retrieving Data From An XML Datasource

This chapter shows how to retrieve XML data from a standard data source. Such source can be a file, an HTTP object or a text string. The method described in this chapter is the simplest way to retrieve XML data. More advanced ways are described in the next chapters.

A Very Simple Example

This section describes a very simple XML application. It parses XML data from a stream and dumps it "pretty-printed" to the standard output. While its use is very limited, it shows how to set up a parser and parse an XML document.

```java
import net.n3.nanoxml.*;  //1
import java.io.*;

public class DumpXML
{
    public static void main(String[] args)
        throws Exception
    {
        IXMLParser parser = XMLParserFactory
    }  //2
```
Retrieving Data From An XML Datasource

```java
.createDefaultXMLParser();
IXMLReader reader = StdXMLReader.fileReader("test.xml");
parser.setReader(reader);
IXMLElement xml = (IXMLElement) parser.parse();
XMLWriter writer = new XMLWriter(System.out);
writer.write(xml);
}
```

(1) The NanoXML classes are located in the package net.n3.nanoxml.

(2) This command creates an XML parser. The actual class of the parser is dependent on the value of the system property net.n3.nanoxml.XMLParser, which is by default net.n3.nanoxml.StdXMLParser.

(3) The command creates a "standard" reader which reads its data from the file called test.xml.

Usually you can use StdXMLReader to feed the XML data to the parser. The default reader is able to set up HTTP connections when retrieving DTDs or entities from different machines. If necessary, you can supply your own reader to e.g. provide support for PUBLIC identifiers.

(4) The XML parser now parses the data read from test.xml and creates a tree of parsed XML elements.

The structure of those elements will be described in the next section.

(5) An XMLWriter can be used to dump a "pretty-printed" view of the parsed XML data on an output stream. In this case, we dump the read data to the standard output (System.out).

Analyzing The Data

You can easily traverse the logical tree generated by the parser. If you need to create your own object tree, you can create your custom builder, which is described in chapter 3.

The default XML builder, StdXMLBuilder generates a tree of IXMLElement objects. Every such object has a name and can have attributes, #PCDATA content and child objects.
The following XML data:

```xml
<FOO attr1="fred" attr2="barney">
  <BAR a1="flintstone" a2="rubble">
    Some data.
  </BAR>
  <QUUX/>
</FOO>
```

is parsed to the following objects:

- **Element FOO**:
  - Attributes = { "attr1"="fred", "attr2"="barney" }
  - Children = { BAR, QUUX }
  - PCData = null

- **Element BAR**:
  - Attributes = { "a1"="flintstone", "a2"="rubble" }
  - Children = {}
  - PCData = "Some data."

- **Element QUUX**:
  - Attributes = {}
  - Children = {}
  - PCData = null

You can retrieve the name of an element using the method `getFullName`, thus:

```java
FOO.getFullName() \rightarrow "FOO"
```

You can enumerate the attribute keys using the method `enumerateAttributeNames`:

```java
Enumeration enum = FOO.enumerateAttributeNames();
while (enum.hasMoreElements()) {
    System.out.print(enum.nextElement());
    System.out.print(' ');
}
\rightarrow attr1 attr2
```
Retrieving Data From An XML Datasource

You can retrieve the value of an attribute using getAttribute:

```java
FOO.getAttribute("attr1", null) -> "fred"
```

The child elements can be enumerated using the method enumerateChildren:

```java
Enumeration enum = FOO.enumerateChildren();
while (enum.hasMoreElements()) {
    System.out.print(enum.nextElement() + ' ');
}
```

→ BAR QUUX

If the element contains parsed character data (#PCDATA) as its only child. You can retrieve that data using `getContent`:

```java
BAR.getContent() -> "Some data."
```

If an element contains both #PCDATA and XML elements as its children, the character data segments will be put in untitled XML elements (whose name is null).

`IXMLElement` contains many convenience methods for retrieving data and traversing the XML tree. You can find them on page 78.

Generating XML

You can very easily create a tree of XML elements or modify an existing one.

To create a new tree, just create an `IXMLElement` object:

```java
IXMLElement elt = new XML.Element("ElementName");
```

You can add an attribute to the element by calling `setAttribute`.

```java
elt.setAttribute("key", "value");
```

You can add a child element to an element by calling `addChild`:
Namespaces

```java
IXMLElement child = elt.createElement("Child");
elt.addChild(child);
```

Note that the child element is created calling the method `createElement`. This insures that the child instance is compatible with its new parent.

If an element has no children, you can add `#PCDATA` content to it using `setContent`:

```java
child.setContent("Some content");
```

If the element does have children, you can add `#PCDATA` content to it by adding an untitled element, which you create by calling `createPCDataElement`:

```java
IXMLElement pcdata = elt.createPCDataElement();
pcdata.setContent("Blah blah");
elt.addChild(pcdata);
```

When you have created or edited the XML element tree, you can write it out to an output stream or writer using an `XMLWriter`:

```java
java.io.Writer output = ...;
IXMLElement xmltree = ...;
XMLWriter xmlwriter = new XMLWriter(output);
writer.write(xmltree);
```

---

**Namespaces**

As of version 2.1, NanoXML has support for namespaces. Namespaces allow you to attach a URI to the name of an element name or an attribute. This URI allows you to make a distinction between similarly named entities coming from different sources. More information about namespaces can be found in the XML Namespaces recommendation, which can be found at [http://www.w3c.org/TR/REC-xml-names](http://www.w3c.org/TR/REC-xml-names).

Please note that a DTD has no support for namespaces. It is important to understand that an XML document can have only one DTD. Though the namespace URI is often presented as a URL, that URL is not a system ID for a DTD. The only function of a namespace URI is to provide a globally unique name.
As an example, let’s have the following XML data:

```xml
  <chapter xmlns="http://nanoxml.n3.net/chapter"
    title="Introduction" doc:id="chapter1"/>
</doc:book>
```

The doc:book top-level element uses the namespace "http://nanoxml.n3.net/book". The prefix is used as an alias for the namespace, which is defined in the attribute xmlns:doc. This prefix is defined for the doc:book element and its child elements.

The chapter element uses the namespace "http://nanoxml.n3.net/chapter". Because the namespace URI has been defined as the value of the xmlns attribute, the namespace is the default namespace for the chapter element. Default namespaces are inherited by the child elements, but only for their names. Attributes never have a default namespace.

The chapter element has an attribute doc:id, which is defined in the same namespace as doc:book because of the doc prefix.

NanoXML 2.1 offers some variants on the standard retrieval methods to allow the application to access the namespace information.

In the following examples, we assume the variable `book` to contain the doc:book element and the variable `chapter` to contain the chapter element.

To get the full name, which includes the namespace prefix, of the element, use `getFullName`:

```java
book.getFullName() → "doc:book"
chapter.getFullName() → "chapter"
```

To get the short name, which excludes the namespace prefix, of the element, use `getName`:

```java
book.getName() → "book"
chapter.getName() → "chapter"
```

For elements that have no associated namespace, `getName` and `getFullName` are equivalent.
To get the namespace URI associated with the name of the element, use `getNamespace`:

```java
chapter.getNamespace()
   → "http://nanoxml.n3.net/chapter"
```

If no namespace is associated with the name of the element, this method returns `null`.

You can get an attribute of an element using either its full name (which includes its prefix) or its short name together with its namespace URI, so the following two instructions are equivalent:

```java
chapter.getAttribute("doc:id", null)
chapter.getAttribute("id",
```

Note that the title attribute of `chapter` has no namespace, even though the `chapter` element name has a default namespace.

You can create a new element which uses a namespace this way:

```java
   "http://nanoxml.n3.net/book");
chapter = book.createElement("chapter",
   "http://nanoxml.n3.net/chapter");
```

You can add an attribute which uses a namespace this way:

```java
chapter.setAttribute("doc:id",
   "http://nanoxml.n3.net/book",
   chapterId);
```
Retrieving Data From An XML Datasource
If you're retrieving data from a stream, but you don't want to wait to process the data until it's completely read, you can use streaming.

The XML Builder

The XML data tree is created using an XML builder. By default, the builder creates a tree of IXMLElement.

While the parser parses the data, it notifies the builder of any elements it encounters. Using this information, the builder generate the object tree. When the parser is done processing the data, it retrieves the object tree from the builder using getResult.

The following example shows a simple builder that prints the notifications on the standard output.

```java
import java.io.*;
import net.n3.nanoxml.*;

public class MyBuilder
    implements IXMLBuilder
{
```
public void startBuilding(String systemID,  //1  
    int lineNr)
{
    System.out.println("Document started");
}

public void  //2  
    newProcessingInstruction(String target,  
    Reader reader)
    throws IOException
{
    System.out.println("New PI with target "+ target);
}

public void startElement(String name,  //3  
    String nsPrefix,  
    String nsSystemID,  
    String systemID,  
    int lineNr)
{
    System.out.println("Element started: "+ name);
}

public void endElement(String name,  //4  
    String nsPrefix,  
    String nsSystemID)
{
    System.out.println("Element ended: " + name);
}

public void addAttribute(String key,  //5  
    String nsPrefix,  
    String nsSystemID,  
    String value,  
    String type)
{
    System.out.println("  " + key + ": " + type + ": " + value);
}

public void elementAttributesProcessed( //6  
    String name,  
    String nsPrefix,  
    String nsSystemID)
Registering an XML Builder

You can register the builder to the parser using the method `setBuilder`.

```java
{  // nothing to do
}

public void addPCData(Reader reader,             //7
        String systemID,
        int    lineNr)
    throws IOException
{
    System.out.println("#PCDATA");
}

public Object getResult()                        //8
{  //8
    return null;
}
}
```

(1) The XML parser started parsing the document. The `lineNr` parameter contains the line number where the document starts.

(2) The XML parser encountered a processing instruction (PI) which is not handled by the parser itself. The `target` contains the target of the PI. The contents of the PI can be read from `reader`.

(3) A new element has been started at line `lineNr`. The name of the element is `name`.

(4) The current element has ended. For convenience, the name of that element is put in the parameter `name`.

(5) An attribute is added to the current element.

(6) This method is called when all the attributes of the current element have been processed.

(7) A `#PCDATA` section has been encountered. The contents of the section can be read from `reader`.

(8) This method is called when the parsing has finished. If the builder has a result, it has to return it to the parser in this method.
The following example shows how to create a parser which uses the builder we created in the previous section:

```java
import net.n3.nanoxml.*;
import java.io.*;

public class DumpXML
{
    public static void main(String args[])
    throws Exception
    {
        IXMLParser parser = XMLParserFactory.createDefaultXMLParser();
        IXMLReader reader = StdXMLReader.fileReader("test.xml");
        parser.setReader(reader);
        parser.setBuilder(new MyBuilder());
        parser.parse();
    }
}
```
This chapter explains how you can customize the NanoXML parser setup. Unlike NanoXML 1, NanoXML/Java 2 is designed as a framework: it is composed of many different components which you can plug together. It’s possible to change the reader, the builder, the validator and even the parser.

The following figure gives a short representation of the major components.

The **reader** retrieves data from a Java input stream and provides character data to the other components.

The **parser** converts the character data it retrieves from the reader to XML events which it sends to the builder.

The **validator** parses a DTD and validates the XML data. The current validator does only the minimum necessary for a non-validating parser.

The **entity resolvers** converts entity references (&...;) and parameter entity references (%...;) to character data. The resolver uses the reader to access external entities.

The **builder** interpretes XML events coming from the parser and builds a tree of XML elements. The standard builder creates a tree of `IXMLElement`. You can provide your own builder to create a custom tree or if you are interested in the XML events themselves, e.g. to use XML streaming.
The NanoXML Reader

The reader retrieves data from some source and feeds it to the other components.

The reader is basically a stack of push-back readers. Every time a new data stream becomes active, the current reader is pushed on a stack. When the current reader has no more data left, the parent reader is popped from the stack.

If you want to implement public IDs using e.g. a catalog file similar to SGML, you could implement a reader by overriding the method `openStream` of `StdXMLReader`:

```java
public class MyReader extends StdXMLReader {
    private Properties publicIDs;

    public MyReader(Properties publicIDs) {
        this.publicIDs = publicIDs;
    }

    public Reader openStream(String publicID, String systemID) throws MalformedURLException, FileNotFoundException, IOException {
        if (publicID != null) {
            systemID = publicIDs.getProperty(publicID, systemID);
        }
        return super.openStream(publicID, systemID);
    }
}
```

In this example, you have to provide a properties object which maps public IDs to system IDs.
Advanced Topics

The NanoXML Parser

The parser analyzes the character stream it retrieves from the reader and sends XML events to the builder. It uses a validator to validate the data and an entity resolver to resolve general entities. You rarely need to create a custom parser. If you need to, you have to implement IXMLParser.

The NanoXML Validator

The validator parses the DTD and checks the XML data. NanoXML 2.0 uses a NonValidator implementation that only performs the minimum necessary for a non-validating parser.

As a DTD is very vague, you can implement your own validator to perform a more fine-grained check of the XML data. The easiest way to create your own validator is to create a subclass of ValidatorPlugin.

The following example shows how to implement a validator. It checks that every attribute named “id” starts with three capital letters.

```java
public class MyValidator
    extends ValidatorPlugin
{
    public void attributeAdded(String key,
                                    String value,
                                    String systemID,
                                    int    lineNr)
    {
        boolean valid = true;
        if (key.equals("id")) {
            if (value.length() < 3) {
                valid = false;
            } else {
                for (int i = 0; i < 3; i++) {
                    char ch = value.charAt(i);
                    if ((ch < 'A') || (ch > 'Z')) {
                        valid = false;
                    }
                }
            }
        }
    }
}
```
The NanoXML Entity Resolvers

```java
if (valid) {
    super.attributeAdded(key, value,
                         systemID, lineNr);
} else {
    this.attributeWithInvalidValue(systemID, 
                                   lineNr, null, key, value);
}
```

To register the validator to a parser, use the following code:

```java
IXMLParser parser ... 
... 
IXMLValidator val1 = parser.getValidator(); 
MyValidator val2 = new MyValidator(); 
val2.setDelegate(val1); 
parser.setValidator(val2); 
```

The NanoXML Entity Resolvers

The entity resolver converts entity references to XML data. If you want e.g. to retrieve entity values from a database, you have to create your own resolver.

Entity resolvers have to implement IXMLEntityResolver. Usually, you only have to make a subclass of XMLEntityResolver and implement the method getEntity or openExternalEntity.

Entities can be used in the XML data and in the DTD. As these entities are independent of each other, there are two entity resolvers.

Standard Entities

The resolver for standard entities has to be registered to the parser by calling setResolver. The following example registers a resolver that forces the entity "&foo;" to be resolved to "bar":

```java
import net.n3.nanoxml.*;
import java.io.*;
```
**Advanced Topics**

```java
class MyResolver extends XMLEntityResolver {
    public Reader getEntity(IXMLReader xmlReader, String name) throws XMLParseException {
        if (name.equals("foo")) {
            return new StringReader("bar");
        } else {
            return super.getEntity(xmlReader, name);
        }
    }
}

public class Demo {
    public static void main(String[] args) throws Exception {
        IXMLParser parser = XMLParserFactory.createDefaultXMLParser();
        parser.setResolver(new MyResolver());
        IXMLReader reader = StdXMLReader.fileReader("test.xml");
        parser.setReader(reader);
        IXMLElement xml = (IXMLElement) parser.parse();
        XMLWriter writer = new XMLWriter(System.out);
        writer.write(xml);
    }
}
```

**Parameter Entities**

Parameter entities have to be registered to the validator by calling `setParameterEntityResolver`. The following example shows a custom version of the Demo class that registers MyResolver as a parameter entity resolver.

```java
public class Demo {
    public static void main(String[] args) throws Exception {
        IXMLParser parser = XMLParserFactory.createDefaultXMLParser();
        parser.setParameterEntityResolver(new MyResolver());
        IXMLReader reader = StdXMLReader.fileReader("test.xml");
        parser.setReader(reader);
        IXMLElement xml = (IXMLElement) parser.parse();
        XMLWriter writer = new XMLWriter(System.out);
        writer.write(xml);
    }
}
```
The NanoXML Builder

The NanoXML Builder interpretes XML events coming from the parser and builds a tree of Java objects. When the parsing is done, the builder hands over its result to the parser.

As explained in chapter 3, the builder can also be used to read XML data while it's being streamed. This feature is useful if you don’t want to wait until all the data has been read before processing the information.

As an example, we have the following XML structure (`document.dtd`):

```xml
<!ELEMENT Chapter (Paragraph*)>
<!ATTLIST Chapter
title CDATA #REQUIRED
id CDATA #REQUIRED>
<!ELEMENT Paragraph (#PCDATA)>
<!ATTLIST Paragraph
align (left|center|right) "left" >
```

The elements are put in the Java classes Chapter and Paragraph which, for convenience, extend the following base class:
public class DocumentElement
{
    protected Properties attrs;
    protected Vector children;

    public DocumentElement()
    {
        this.attrs = new Properties();
        this.children = new Vector();
    }

    public void setAttribute(String attrName, String value)
    {
        this.attrs.put(attrName, value);
    }

    public void addChild(DocumentElement elt)
    {
        this.children.addElement(elt);
    }
}

This base class simply makes it easy for our builder to set attributes and to add children to an element.

The Chapter and Paragraph classes extend this base class to give more practical access to their attributes and children:

public class Chapter
    extends DocumentElement
{
    public String getTitle()
    {
        return this.attrs.getProperty("title");
    }

    public String getID()
    {
        return this.attrs.getProperty("id");
    }

    public Enumeration getParagraphs()
    {

The NanoXML Builder

```java
public class Paragraph
    extends DocumentElement
{
    public static final int LEFT = 0;
    public static final int CENTER = 1;
    public static final int RIGHT = 2;

    private static Hashtable alignments;
    static {
        alignments = new Hashtable();
        alignments.put("left", new Integer(LEFT));
        alignments.put("center", new Integer(CENTER));
        alignments.put("right", new Integer(RIGHT));
    }

    public String getContent()
    {
        return this.attrs.getProperty("#PCDATA");
    }

    public int getAlignment()
    {
        String str = this.attrs.getProperty("align");
        Integer align = alignments.get(str);
        return align.intValue();
    }
}
```

The builder creates the data structure based on the XML events it receives from the parser. Because both Chapter and Paragraph extend DocumentElement, the builder is fairly simple.

```java
import net.n3.nanoxml.*;
import java.util.*;
import java.io.*;

public class DocumentBuilder
    implements IXMLBuilder
{
```

NanoXML/Java 2.1

25
private static Hashtable classes;
private Stack elements;
private DocumentElement topElement;

static {
    classes = new Hashtable();
    classes.put("Chapter", Chapter.class);
    classes.put("Paragraph", Paragraph.class);
}

public void startBuilding(String systemID, int lineNr) {
    this.elements = new Stack();
    this.topElement = null;
}

public void newProcessingInstruction(String target, Reader reader) {
    // nothing to do
}

public void startElement(String name, String nsPrefix, String nsSystemID, String systemID, int lineNr) {
    DocumentElement elt = null;
    try {
        Class cls = (Class) classes.get(name);
        elt = (DocumentElement) cls.newInstance();
    } catch (Exception e) {
        // ignore the exception
    }
    this.elements.push(elt);
    if (this.topElement == null) {
        this.topElement = elt;
    }
}

public void endElement(String name, String nsPrefix, String nsSystemID)
{ DocumentElement child
    = (DocumentElement) this.elements.pop();
if (! this.elements.isEmpty()) {
    DocumentElement parent
        = (DocumentElement) this.elements.peek();
    parent.addChild(child);
}
}

public void addAttribute(String key,
    String nsPrefix,
    String nsSystemID,
    String value,
    String type)
{
    DocumentElement child
        = (DocumentElement) this.elements.peek();
    child.setAttribute(key, value);
}

public void elementAttributesProcessed(
    String name,
    String nsPrefix,
    String nsSystemID)
{
    // nothing to do
}

public void addPCData(Reader reader,
    String systemID,
    int lineNr)
throws IOException
{
    StringBuffer str = new StringBuffer(1024);
    char[] buf = new char[bufSize];
    for (; ; ) {
        int size = reader.read(buf);
        if (size < 0) {
            break;
        }
        str.append(buf, 0, size);
    }
    this.addAttribute("#PCDATA", null, null,
        str.toString(), "CDATA");
Note that, for simplicity, error and exception handling is not present in this example. The builder holds a stack of the current elements it builds. Character data is read from a reader. The method addPCData reads this data in blocks of 1K.

Finally, this application sets up the NanoXML parser and converts an XML document to HTML which it dumps on the standard output:

```java
import java.util.*;
import net.n3.nanoxml. *

public class XML2HTML
{
    public static void main(String[] params)
        throws Exception
    {
        IXMLBuilder builder
            = new DocumentBuilder();
        IXMLParser parser
            = XMLParserFactory
            .createDefaultXMLParser();
        parser.setBuilder(builder);
        IXMLReader reader
            = StdXMLReader.fileReader(param[0]);
        parser.setReader(reader);
        Chapter chapter = (Chapter) parser.parse();
        System.out.println("<!DOCTYPE ... >");
        System.out.print("<HTML><HEAD><TITLE>");
        System.out.println(chapter.getTitle());
        System.out.println("</TITLE></HEAD><BODY>");
        System.out.print("<H1>");
        System.out.println(chapter.getTitle());
        System.out.println("</H1>");
        Enumeration enum = chapter.getParagraphs();
        while (enum.hasMoreElements()) {
            Paragraph para
                = (Paragraph) enum.nextElement();
```
System.out.print("<P>");
System.out.print(para.getContent());
System.out.println("</P>");
}
System.out.println("</BODY></HTML>");
}

If we run the example on the following XML file:

```
<!DOCTYPE Chapter SYSTEM "document.dtd">
<Chapter id="ch01" title="The Title">
  <Paragraph>First paragraph...</Paragraph>
  <Paragraph>Second paragraph...</Paragraph>
</Chapter>
```

The output will be:

```
<!DOCTYPE HTML PUBLIC '-//W3C//DTD HTML 4.01//EN'
  'http://www.w3.org/TR/html4/strict.dtd'>
<html><head><title>The Title</title></head><body>
<h1>The Title</h1>
<p>First paragraph...</p>
<p>Second paragraph...</p>
</body>
```
This chapter gives an overview of the Java classes.

**net.n3.nanoxml.IXMLBuilder (interface)**

NanoXML uses `IXMLBuilder` to construct the XML data structure it retrieved from its data source. You can supply your own builder or you can use the default builder of NanoXML. You can find more information about custom builders on page 23.

If a method of the builder throws an exception, the parsing is aborted and the `parse` method throws an `XMLException` which encapsulates the original exception.
addAttribute

```java
void addAttribute(java.lang.String key,
                 java.lang.String nsPrefix,
                 java.lang.String nsURI,
                 java.lang.String value,
                 java.lang.String type)
throws Exception;
```

This method is called when a new attribute of an XML element is encountered.

- **key**: The key (name) of the attribute.
- **nsPrefix**: The prefix used to identify the namespace. If no namespace has been specified, this parameter is null.
- **nsURI**: The URI associated with the namespace. If no namespace has been specified, or no URI is associated with nsPrefix, this parameter is null.
- **value**: The value of the attribute.
- **type**: The type of the attribute. If no type is known, "CDATA" is returned.

addPCData

```java
void addPCData(java.io.Reader reader,
               java.lang.String systemID,
               int lineNr)
throws Exception;
```

This method is called when a PCDATA element is encountered. A Java reader is supplied from which you can read the data. The reader will only read the data of the element. You don’t need to check for boundaries. If you don’t read the full element, the rest of the data is skipped. You also don’t have to care about entities: they are resolved by the parser.

- **reader**: The method can retrieve the data from this reader. You may close the reader before reading all its data and you cannot read too much data.
- **systemID**: The system ID of the XML data source
- **lineNr**: The line in the source where the element starts.
null

net.n3.nanoxml.IXMLBuilder (interface)

elementAttributesProcessed

```java
void elementAttributesProcessed(java.lang.String name,
                                  java.lang.String nsPrefix,
                                  java.lang.String nsURI)
  throws Exception;
```

This method is called when all the attributes of an XML element have been processed.

- **name**: The name of the element.
- **nsPrefix**: The prefix used to identify the namespace. If no namespace has been specified, this parameter is `null`.
- **nsURI**: The URI associated with the namespace. If no namespace has been specified, or no URI is associated with nsPrefix, this parameter is `null`.

endElement

```java
void endElement(java.lang.String name,
                 java.lang.String nsPrefix,
                 java.lang.String nsURI)
  throws Exception;
```

This method is called when the end of an XML element is encountered.

- **name**: The name of the element.
- **nsPrefix**: The prefix used to identify the namespace. If no namespace has been specified, this parameter is `null`.
- **nsURI**: The URI associated with the namespace. If no namespace has been specified, or no URI is associated with nsPrefix, this parameter is `null`.
**getResult**

```java
java.lang.Object getResult()
throws Exception;
```

Returns the result of the building process. This method is called just before the `parse` method of `IXMLParser` returns. The method has to return the result of the building process.

**newProcessingInstruction**

```java
void newProcessingInstruction(java.lang.String target,
java.io.Reader reader)
throws Exception;
```

This method is called when a processing instruction is encountered. A PI with a reserved target ("xml" with any case) is never reported.

- **target** The processing instruction target.
- **reader** The method can retrieve the parameter of the PI from this reader. You may close the reader before reading all its data and you cannot read too much data.

**startBuilding**

```java
void startBuilding(java.lang.String systemID,
int lineNr)
throws Exception;
```

This method is called before the parser starts processing its input.

- **systemID** The system ID of the XML data source
- **lineNr** The line on which the parsing starts.
**startElement**

```java
void startElement(String name,
                 String nsPrefix,
                 String nsURI,
                 String systemID,
                 int lineNr)
    throws Exception;
```

This method is called when a new XML element is encountered.

- **name**
  - The name of the element.

- **nsPrefix**
  - The prefix used to identify the namespace. If no namespace has been specified, this parameter is `null`.

- **nsURI**
  - The URI associated with the namespace. If no namespace has been specified, or no URI is associated with `nsPrefix`, this parameter is `null`.

- **systemID**
  - The system ID of the XML data source.

- **lineNr**
  - The line in the source where the element starts.
**net.n3.nanoxml.IXMLElement (interface)**

```java
<<interface>>
IXMLElement
```

**addChild**

```java
void addChild(IXMLElement child);
```

Adds a child element.

child The child to add.

**createElement**

```java
void createElement(java.lang.String fullName);
```

Creates a new empty element that is compatible with the receiver.

fullName The full name of the element.

```java
void createElement(java.lang.String fullName,
java.lang.String systemID,
int lineNr);
```

Creates a new empty element that is compatible with the receiver.

fullName The full name of the element.
systemID The system ID of the XML data where the element starts.
lineNr The line in the XML data where the element starts.
void createElement(java.lang.String fullName,
             java.lang.String namespace);

Creates a new empty element that is compatible with the receiver.

fullName The full name of the element.
namespace The namespace URI

void createElement(java.lang.String fullName,
                    java.lang.String namespace
                    java.lang.String systemID,
                    int lineNr);

Creates a new empty element that is compatible with the receiver.

fullName The full name of the element.
namespace The namespace URI
systemID The system ID of the XML data where the element starts
lineNr The line in the XML data where the element starts

createPCDataElement
IXMLElement createPCDataElement();

Creates an element that can be used for #PCDATA content.

enumerateAttributeNames
java.util.Enumeration enumerateAttributeNames();

Returns an enumeration of all attribute names.

enumerateChildren
java.util.Enumeration enumerateChildren();

Returns an enumeration of all child elements.
equals

boolean equals(java.lang.Object obj);

Returns true if the element is equal to obj. If obj cannot be casted to an IXMLElement, the method returns false.

obj The object.

equalsXMLElement

boolean equalsXMLElement(IXMLElement elt);

Returns true if the element is equal to elt. This method is faster than the more generic equals method.

elt The other element.

getAttribute

java.lang.String getAttribute(java.lang.String name,
java.lang.String defaultValue);

int getAttribute(java.lang.String name,
int defaultValue);

Returns the value of an attribute. If the attribute is not present, defaultValue is returned.

name The name of the attribute.
defaultValue The default value of the attribute.
net.n3.nanoxml.IXMLElement (interface)

    java.lang.String getAttribute(java.lang.String name,
            java.lang.String namespace,
            java.lang.String defaultValue);
    int getAttribute(java.lang.String name,
            java.lang.String namespace,
            int defaultValue);

Returns the value of an attribute. The attribute is specified using its short name and
the associated namespace URI. If the attribute is not present, defaultValue is
returned.

    name            The name of the attribute.
    namespace       The namespace URI, or null if no namespace is
                    associated.
    defaultValue    The default value of the attribute.

getAttributeCount

    int getAttributeCount();

Returns the number of attributes.

getAttributeNamespace

    java.lang.String getAttributeNamespace(
            java.lang.String name);

Returns the namespace URI associated with an attribute. You have to specify the
full name of the attribute (i.e. the attribute with its prefix). If no namespace URI is
associated with the attribute, null is returned.

    name            The full name of the attribute.

getAttributes

    java.util.Properties getAttributes();

Returns a Properties object containing all the attributes of the element. The
attributes will be specified using their full name.
getAttributesInNamespace

    java.util.Properties getAttributesInNamespace(
            java.lang.String namespace);

Returns a Properties object containing all the attributes which are associated with
namespace of the element. The attributes will be specified using their short names.

namespace The namespace URI.

getAttributeType

    java.lang.String getAttributeType(
            java.lang.String name);

Returns the type of the attribute.

name The full name of the attribute.

getAttributeType

    java.lang.String getAttributeType(
            java.lang.String name,
            java.lang.String namespace);

Returns the type of the attribute.

name The short name of the attribute.
namespace The namespace URI

getChildAtIndex

    IXMLElement getChildAtIndex(int index);

Returns the child element located at a certain index.

index The index of the child, where the first child has
index 0.
getChildren

    java.util.Vector getChildren();

Returns a vector containing all the child elements. You should not modify the vector.

getChildrenCount

    int getChildrenCount();

Returns the number of children.

getChildrenNamed

    java.util.Vector getChildrenNamed(java.lang.String name);

Returns a vector of all child elements named name.

    name The full name of the children to search.

    java.util.Vector getChildrenNamed(
        java.lang.String name,
        java.lang.String namespace);

Returns a vector of all child elements named name.

    name The short name of the children to search.
    namespace The namespace URI.

getContent

    java.lang.String getContent();

Returns the the #PCDATA content of the element. If the element has a combination of #PCDATA content and child elements, the #PCDATA sections can be retrieved as unnamed child objects. In this case, this method returns null.
**getFirstChildNamed**

```java
IXMLElement getFirstChildNamed(java.lang.String name);
```

Returns the first child named `name`. If there is no such child, `null` is returned.

```java
name                     The full name of the child to search.
```

```java
IXMLElement getFirstChildNamed(
    java.lang.String name,
    java.lang.String namespace);
```

Returns the first child named `name`. If there is no such child, `null` is returned.

```java
name                     The full name of the child to search.
namespace                The namespace URI.
```

**getFullName**

```java
java.lang.String getFullName();
```

Returns the full name of the element, which is the name including the namespace prefix. If the element only contains `#PCDATA`, this method return `null`.

**getLineNr**

```java
int getLineNr();
```

Returns the line number in the data where the element started.

**getName**

```java
java.lang.String getName();
```

Returns the name of the element, or null if the element only contains `#PCDATA`. 
getNamespace

java.lang.String getNamespace();

Returns the namespace URI of the element, or null if the element only contains #PCDATA or has no namespace associated with it.

getParent

IXMLElement getParent();

Returns the parent element. If the element is the root element, the method returns null.

getSystemID

java.lang.String getSystemID();

Returns the system ID of the data where the element started.

hasAttribute

boolean hasAttribute(java.lang.String name);

Returns true if the element has an attribute named name.

name The full name of the attribute.

boolean hasAttribute(java.lang.String name, java.lang.String namespace);

Returns true if the element has an attribute named name associated with the namespace URI namespace.

name The short name of the attribute.

namespace The namespace URI.
hasChildren
    boolean hasChildren();
    Returns true if the element has children.

isLeaf
    boolean isLeaf();
    Returns true if the element has no children.

removeAttribute
    void removeAttribute(java.lang.String name);
    Removes a attribute.

    name The full name of the attribute.
    void removeAttribute(java.lang.String name,
                          java.lang.String namespace);
    Removes a attribute.

    name The short name of the attribute.
    namespace The namespace URI.

removeChild
    void removeChild(IXMLElement child);
    Removes a child element.

    child The child to add.
**removeChildAtIndex**

```java
void removeChildAtIndex(int index);
```

Removes the child element located at a certain index.

- **index** The index of the child, where the first child has index 0.

**setAttribute**

```java
void setAttribute(java.lang.String name,
                 java.lang.String value);
```

Sets the value of an attribute.

- **name** The full name of the attribute.
- **value** The value of the attribute.

```java
void setAttribute(java.lang.String name,
                 java.lang.String namespace,
                 java.lang.String value);
```

Sets the value of an attribute.

- **name** The name of the attribute.
- **namespace** The namespace URI.
- **value** The value of the attribute.

**setContent**

```java
void setContent(java.lang.String content);
```

Sets the #PCDATA content. It is an error to call this method with a non-null value if there are child objects.

- **content** The (possible null) content.
**setName**

```java
void setName(java.lang.String name);
```

Sets the name of the element.

- **name**  The name of the element.

```java
void setName(java.lang.String name,
             java.lang.String namespace);
```

Sets the name of the element.

- **name**  The full name of the element.
- **namespace**  The namespace URI.

---

**net.n3.nanoxml.IXMLEntityResolver (interface)**

An IXMLEntityResolver resolves entities. More information about custom entity resolvers can be found on page 21.
addExternalEntity

```java
void addExternalEntity(String name, String publicID, String systemID);
```

Adds an external entity.

- **name**: The name of the entity.
- **publicID**: The public ID of the entity, which may be null.
- **systemID**: The system ID of the entity.

addInternalEntity

```java
void addInternalEntity(String name, String value);
```

Adds an internal entity.

- **name**: The name of the entity.
- **value**: The value of the entity.

gentity

```java
java.io.Reader getEntity(IXMLReader xmlReader, String name);
```

Returns a Java reader containing the value of an entity. If the entity could not be resolved, null is returned. The method may throw an XMLParseException if necessary.

- **xmlReader**: The current NanoXML reader.
- **name**: The name of the entity.
**net.n3.nanoxml.IXMLParser (interface)**

`IXMLParser` is the core parser of NanoXML.

**getBuilder**

```java
IXMLBuilder getBuilder();
```

Returns the builder which creates the local structure of the XML data.

**getReader**

```java
IXMLReader getReader();
```

Returns the reader from which the parser retrieves its data.

**getResolver**

```java
IXMLEntityResolver getResolver();
```

Returns the entity resolver.

**getValidator**

```java
IXMLValidator getValidator();
```

Returns the validator that validates the XML data.
parse

java.lang.Object parse()
    throws XMLException;

Parses the data and lets the builder create the logical structure. The method returns
the result of getRoot of the builder described on page 34. If an error occurred
while reading or parsing the data, the method may throw an XMLException.

setBuilder

void setBuilder(IXMLBuilder builder);

Sets the builder which creates the local structure of the XML data.

    builder       The builder

setReader

void setReader(IXMLReader reader);

Sets the reader from which the parser retrieves its data.

    reader       The reader

setResolver

void setResolver(IXMLEntityResolver resolver);

Sets the entity resolver.

    resolver     The resolver
**setValidator**

```java
void setValidator(IXMLValidator validator);
```

Sets the validator that validates the XML data.

- **validator** The validator

---

**net.n3.nanoxml.IXMLReader (interface)**

```
<<interface>>

IXMLReader

StdXMLReader
```

**IXMLReader** reads the data to be parsed. More information about custom readers can be found on page 19.

---

**atEOF**

```java
boolean atEOF()
    throws java.io.IOException;
```

Returns `true` if there are no more characters left to be read.

---

**atEOFOfCurrentStream**

```java
boolean atEOFOfCurrentStream()
    throws java.io.IOException;
```

Returns `true` if the current stream has no more characters left to be read.
**getLineNr**

```java
int getLineNr();
```

Returns the line number of the data in the current stream.

**getPublicID**

```java
java.lang.String getPublicID();
```

Returns the public ID of the current stream.

**getSystemID**

```java
java.lang.String getSystemID();
```

Returns the system ID of the current stream.

**openStream**

```java
java.io.Reader openStream(java.lang.String publicID,
                          java.lang.String systemID)
                      throws java.net.MalformedURLException,
                          java.io.FileNotFoundException,
                          java.io.IOException;
```

Opens a stream from a public and system ID.

- **publicID** The public ID of the entity (may be null)
- **systemID** The system ID of the entity.

**read**

```java
char read()
           throws java.io.IOException;
```

Reads a character.
**setPublicID**

`void setPublicID(java.lang.String publicID);`

Sets the public ID of the current stream.

- **publicID**: The public ID.

**setSystemID**

`void setSystemID(java.lang.String systemID)`

Throws `java.net.MalformedURLException;`

Sets the system ID of the current stream.

- **systemID**: The system ID.

**startNewStream**

`void startNewStream(java.io.Reader reader);`

Starts a new stream from a Java reader. The new stream is used temporarily to read data from. If that stream is exhausted, control returns to the "parent" stream.

- **reader**: The reader to read the new data from.

**unread**

`void unread(char ch)`

Throws `java.io.IOException`

Pushes the last character read back to the stream.

- **ch**: The character to push back.
net.n3.nanoxml.IXMLValidator (interface)

IXMLValidator processes the DTD and handles entity references. More information about custom validators can be found on page 20.

attributeAdded

```java
void attributeAdded(java.lang.String key,
                     java.lang.String value,
                     java.lang.String systemID,
                     int lineNr)
```

throws java.lang.Exception;

Indicates that an attribute has been added to the current element.

key The name of the attribute.
value The value of the attribute
systemID The system ID of the XML data of the element
lineNr The line number in the XML data of the element

elementAttributesProcessed

```java
void elementAttributesProcessed(
    java.lang.String name,
    java.util.Properties extraAttributes,
    java.lang.String systemID,
    int lineNr)
```

throws java.lang.Exception;

This method is called when the attributes of an XML element have been processed.
If there are attributes with a default value which have not been specified yet, they have to be put in `extraAttributes`.

```java
void elementEnded(java.lang.String name, java.lang.String systemID, int lineNr)
    throws java.lang.Exception;
```

Indicates that an element has ended.

```java
void elementStarted(java.lang.String name, java.lang.String systemID, int lineNr)
    throws java.lang.Exception;
```

Indicates that an element has been started.
getParameterEntityResolver

IXMLEntityResolver getParameterEntityResolver();

Returns the parameter entity resolver.

parseDTD

void parseDTD(java.lang.String publicID, IXMLReader reader, IXMLEntityResolver resolver, boolean external) throws java.lang.Exception;

Parses the DTD. The validator object is responsible for reading the full DTD.

publicID The public ID, which may be null.
reader The reader to read the DTD from.
resolver The entity resolver.
external true if the DTD is external

PCDataAdded

void PCDataAdded(java.lang.String systemID, int lineNr) throws java.lang.Exception;

Indicates that a new #PCDATA element has been encountered.

systemID The system ID of the XML data of the element
lineNr The line number in the XML data of the element
setParameterEntityResolver

```java
void setParameterEntityResolver(
    IXMLEntityResolver resolver);
```

Sets the parameter entity resolver.

- **resolver** 
The resolver

---

**net.n3.nanoxml.NonValidator**

NonValidator is a concrete implementation of IXMLValidator which processes the DTD and handles entity definitions. It does not do any validation itself.

**attributeAdded**

```java
void attributeAdded(java.lang.String key,
    java.lang.String value,
    java.lang.String systemID,
    int lineNr)
    throws java.lang.Exception;
```

Indicates that an attribute has been added to the current element.

- **key** 
The name of the attribute.
- **value** 
The value of the attribute
- **systemID** 
The system ID of the XML data of the element
- **lineNr** 
The line number in the XML data of the element
elementAttributesProcessed

```java
void elementAttributesProcessed(
    java.lang.String name,
    java.util.Properties extraAttributes,
    java.lang.String systemID,
    int lineNr)
throws java.lang.Exception;
```

This method is called when the attributes of an XML element have been processed.

If there are attributes with a default value which have not been specified yet, they have to be put in `extraAttributes`.

- name: The name of the element.
- extraAttributes: Where to put extra attributes
- systemID: The system ID of the XML data of the element
- lineNr: The line number in the XML data of the element

elementEnded

```java
void elementEnded(java.lang.String name,
    java.lang.String systemID,
    int lineNr)
throws java.lang.Exception;
```

Indicates that an element has ended.

- name: The name of the element.
- systemID: The system ID of the XML data of the element
- lineNr: The line number in the XML data of the element
elementStarted

```java
void elementStarted(java.lang.String name,
                     java.lang.String systemID,
                     int lineNr)
    throws java.lang.Exception;
```

Indicates that an element has been started.

- **name** The name of the element.
- **systemID** The system ID of the XML data of the element
- **lineNr** The line number in the XML data of the element

getParameterEntityResolver

```java
IXMLEntityResolver getParameterEntityResolver();
```

Returns the parameter entity resolver.

parseDTD

```java
void parseDTD(java.lang.String publicID,
              IXMLReader reader,
              IXMLEntityResolver resolver,
              boolean external)
    throws java.lang.Exception;
```

 Parses the DTD. The validator object is responsible for reading the full DTD.

- **publicID** The public ID, which may be null.
- **reader** The reader to read the DTD from.
- **resolver** The entity resolver.
- **external** true if the DTD is external
PCDataAdded

    void PCDataAdded(java.lang.String systemID,
            int    lineNr)
    throws java.lang.Exception;

Indicates that a new #PCDATA element has been encountered.

    systemID     The system ID of the XML data of the element
    lineNr       The line number in the XML data of the element

setParameterEntityResolver

    void setParameterEntityResolver(
            IXMLEntityResolver resolver);

Sets the parameter entity resolver.

    resolver       The resolver

StdXMLBuilder is a concrete implementation of IXMLBuilder which creates
a tree of IXMLElement from an XML data source.
addAttribute

```java
void addAttribute(java.lang.String key,
                   java.lang.String nsPrefix,
                   java.lang.String nsURI,
                   java.lang.String value,
                   java.lang.String type)

throws Exception;
```

This method is called when a new attribute of an XML element is encountered.

- **key** The key (name) of the attribute.
- **nsPrefix** The prefix used to identify the namespace. If no namespace has been specified, this parameter is `null`.
- **nsURI** The URI associated with the namespace. If no namespace has been specified, or no URI is associated with nsPrefix, this parameter is `null`.
- **value** The value of the attribute.
- **type** The type of the attribute. If no type is known, "CDATA" is returned.

addPCData

```java
void addPCData(java.io.Reader reader,
                java.lang.String systemID,
                int lineNr)

throws Exception;
```

This method is called when a PCDATA element is encountered. A Java reader is supplied from which you can read the data. The reader will only read the data of the element. You don’t need to check for boundaries. If you don’t read the full element, the rest of the data is skipped. You also don’t have to care about entities; they are resolved by the parser.

- **reader** The method can retrieve the parameter of the PI from this reader. You may close the reader before reading all its data and you cannot read too much data.
- **systemID** The system ID of the XML data source.
- **lineNr** The line in the source where the element starts.
**elementAttributesProcessed**

```java
void elementAttributesProcessed(java.lang.String name,
                                java.lang.String nsPrefix,
                                java.lang.String nsURI)
throws Exception;
```

This method is called when all the attributes of an XML element have been processed.

- **name** The name of the element.
- **nsPrefix** The prefix used to identify the namespace. If no namespace has been specified, this parameter is null.
- **nsURI** The URI associated with the namespace. If no namespace has been specified, or no URI is associated with nsPrefix, this parameter is null.

**endElement**

```java
void endElement(java.lang.String name,
                java.lang.String nsPrefix,
                java.lang.String nsURI)
throws Exception;
```

This method is called when the end of an XML element is encountered.

- **name** The name of the element.
- **nsPrefix** The prefix used to identify the namespace. If no namespace has been specified, this parameter is null.
- **nsURI** The URI associated with the namespace. If no namespace has been specified, or no URI is associated with nsPrefix, this parameter is null.
getResult

```java
java.lang.Object getResult()
    throws Exception;
```

Returns the result of the building process. This method is called just before the `parse` method of `IXMLParser` returns. The result is of type `IXMLElement`, but because of limitations of the Java language, I cannot specify that here.

newProcessingInstruction

```java
void newProcessingInstruction(java.lang.String target,
                                java.io.Reader reader)
    throws Exception;
```

This method is called when a processing instruction is encountered. A PI with a reserved target ("xml" with any case) is never reported.

- **target**: The processing instruction target.
- **reader**: The method can retrieve the data from this reader. You may close the reader before reading all its data and you cannot read too much data.

startBuilding

```java
void startBuilding(String systemID,
                    int lineNr)
    throws Exception;
```

This method is called before the parser starts processing its input.

- **systemID**: The system ID of the XML data source
- **lineNr**: The line on which the parsing starts.
**net.n3.nanoxml.StdXMLBuilder**

### startElement

```java
void startElement(java.lang.String name,
                  java.lang.String nsPrefix,
                  java.lang.String nsURI,
                  java.lang.String systemID,
                  int lineNr)
  throws Exception;
```

This method is called when a new XML element is encountered.

- **name**: The name of the element.
- **nsPrefix**: The prefix used to identify the namespace. If no namespace has been specified, this parameter is `null`.
- **nsURI**: The URI associated with the namespace. If no namespace has been specified, or no URI is associated with `nsPrefix`, this parameter is `null`.
- **systemID**: The system ID of the XML data source.
- **lineNr**: The line in the source where the element starts.

### StdXMLBuilder (constructor)

```java
StdXMLBuilder();
```

Creates a builder that creates `XMLElement` instances.

```java
StdXMLBuilder(XMLElement prototype);
```

Creates a builder that creates clones of prototype.

- **prototype**: The prototype element to clone.
net.n3.nanoxml.StdXMLParser

\[
\text{StdXMLParser} \quad \downarrow \\
\text{IXMLParser}
\]

StdXMLParser is the concrete instance of IXMLParser.

**getBuilder**

```java
IXMLBuilder getBuilder();
```

Returns the builder which creates the local structure of the XML data.

**getReader**

```java
IXMLReader getReader();
```

Returns the reader from which the parser retrieves its data.

**getResolver**

```java
IXMLEntityResolver getResolver();
```

Returns the entity resolver.

**getValidator**

```java
IXMLValidator getValidator();
```

Returns the validator that validates the XML data.
parse

    java.lang.Object parse()
    throws XMLException;

Parses the data and lets the builder create the logical structure. The method returns the result of `getResult` of the builder described on page 34. If an error occurred while reading or parsing the data, the method may throw an `XMLException`.

setBuilder

    void setBuilder(IXMLBuilder builder);

Sets the builder which creates the local structure of the XML data.

    builder The builder

setReader

    void setReader(IXMLReader reader);

Sets the reader from which the parser retrieves its data.

    reader The reader

setResolver

    void setResolver(IXMLEntityResolver resolver);

Sets the entity resolver.

    resolver The resolver
**setValidator**

```java
void setValidator(IXMLValidator validator);
```

Sets the validator that validates the XML data.

**validator**

The validator

---

`net.n3.nanoxml.StdXMLReader`

STDXMLReader is the concrete implementation of `IXMLReader` which reads the data to be parsed.

**atEOF**

```java
boolean atEOF() throws java.io.IOException;
```

Returns `true` if there are no more characters left to be read.

**atEOFOfCurrentStream**

```java
boolean atEOFOfCurrentStream() throws java.io.IOException;
```

Returns `true` if the current stream has no more characters left to be read.
fileReader (static)

    static IXMLReader fileReader(java.lang.String filename);

Creates a new reader using a file as input.

    filename
    The name of the file containing the XML data.

getEncoding (protected)

    protected String getEncoding(java.lang.String str);

Retrieves the encoding from a <?xml ... ?> tag. This method is only called when
the encoding could not be determined automatically and when the tag is the very
first element in the data.

    str
    The first PI tag in the XML data.

g.getLineNr

    int getlineNr();

Returns the line number of the data in the current stream.

g.getPublicID

    java.lang.String getPublicID();

Returns the public ID of the current stream.

g.getSystemID

    java.lang.String getSystemID();

Returns the system ID of the current stream.
openStream

```java
java.io.Reader openStream(java.lang.String publicID,
                         java.lang.String systemID)
    throws java.net.MalformedURLException,
            java.io.FileNotFoundException,
            java.io.IOException;
```

Opens a stream from a public and system ID.

- `publicID` The public ID of the entity (may be null)
- `systemID` The system ID of the entity.

read

```java
char read()
    throws java.io.IOException;
```

Reads a character.

setPublicID

```java
void setPublicID(java.lang.String publicID);
```

Sets the public ID of the current stream.

- `publicID` The public ID.

setSystemID

```java
void setSystemID(java.lang.String systemID)
    throws java.net.MalformedURLException;
```

Sets the system ID of the current stream.

- `systemID` The system ID.
**net.n3.nanoxml.StdXMLReader**

### startNewStream

```
void startNewStream(java.io.Reader reader);
```

Starts a new stream from a Java reader. The new stream is used temporary to read data from. If that stream is exhausted, control returns to the "parent" stream.

- **reader**
  - The reader to read the new data from.

### StdXMLReader (constructor)

```
StdXMLReader(java.lang.String publicID,
            java.lang.String systemID)
```

- **throws**
  - `java.net.MalformedURLException`
  - `java.io.FileNotFoundException`
  - `java.io.IOException`

Initializes a reader from a system ID and optional public ID.

- **systemID**
  - The system ID.
- **publicID**
  - The public ID (may be null)

```
StdXMLReader(java.io.Reader reader);
```

Initializes a reader from a Java reader.

- **reader**
  - The Java reader.

```
StdXMLReader(java.io.InputStream stream);
```

Initializes a reader from a Java input stream.

- **stream**
  - The Java input stream.
stream2reader (protected)

```java
protected java.io.Reader stream2reader(
    java.io.InputStream    stream,
    java.lang.StringBuffer charsRead);
```

Converts a Java input stream to a Java reader while detecting the encoding.

- **stream** The input for the XML data.
- **charsRead** Buffer where to put characters that have been read.

stringReader (static)

```java
static IXMLReader stringReader(java.lang.String str);
```

Creates a new reader using a string as input.

- **str** The string containing the XML data.

unread

```java
void unread(char ch)
```

Throws `java.io.IOException`

Pushes the last character read back on the stream.

- **ch** The character to push back
ValidatorPlugin allows the application to insert additional validators into the NanoXML framework. More information about custom validators can be found on page 20.

attributeAdded

```java
void attributeAdded(String key,
    String value,
    String systemID,
    int lineNr)
    throws Exception;
```

Indicates that an attribute has been added.

- **key**: The name of the attribute.
- **value**: The value of the attribute.
- **systemID**: The system ID of the XML data of the element.
- **lineNr**: The line number in the XML data of the element.
**elementStarted**

```java
void elementStarted(java.lang.String name,
                    java.lang.String systemID,
                    int lineNr)
    throws java.lang.Exception;
```

Indicates that an element has been started.

- **name**: The name of the element.
- **systemID**: The system ID of the XML data of the element.
- **lineNr**: The line number in the XML data of the element.

**elementAttributesProcessed**

```java
void elementAttributesProcessed(
    java.lang.String     name,
    java.util.Properties extraAttributes,
    java.lang.String     systemID,
    int lineNr)
    throws java.lang.Exception;
```

This method is called when the attributes of an XML element have been processed.

If there are attributes with a default value which have not been specified yet, they have to be put in `extraAttributes`.

- **name**: The name of the element.
- **extraAttributes**: Where to put extra attributes.
- **systemID**: The system ID of the XML data of the element.
- **lineNr**: The line number in the XML data of the element.

**getDelegate**

```java
IXMLValidator getDelegate();
```

Returns the delegate.
**getParameterEntityResolver**

```java
IXMLEntityResolver getParameterEntityResolver();
```

Returns the parameter entity resolver.

**invalidAttributeValue**

```java
void invalidAttributeValue(java.lang.String systemID, int lineNr, java.lang.String elementName, java.lang.String attributeName, java.lang.String attributeValue)
```

Throws an `XMLValidationException` to indicate that an attribute has an invalid value.

- **systemID** The system ID of the XML data of the element
- **lineNr** The line number in the XML data of the element
- **elementName** The name of the element
- **attributeName** The name of the attribute
- **attributeValue** The value of the attribute

**missingAttribute**

```java
void missingAttribute(java.lang.String systemID, int lineNr, java.lang.String elementName, java.lang.String attributeName)
```

Throws an `XMLValidationException` to indicate that an attribute is missing.

- **systemID** The system ID of the XML data of the element
- **lineNr** The line number in the XML data of the element
- **elementName** The name of the element
- **attributeName** The name of the missing attribute
missingElement

```java
void missingElement(java.lang.String systemID,
                     int lineNr,
                     java.lang.String parentElementName,
                     java.lang.String missingElementName)
    throws XMLValidationException;
```

Throws an XMLValidationException to indicate that an element is missing.

- **systemID**: The system ID of the XML data of the element
- **lineNr**: The line number in the XML data of the element
- **parentElementName**: The name of the parent element
- **missingElementName**: The name of the missing element

missingPCData

```java
void missingPCData(java.lang.String systemID,
                    int lineNr,
                    java.lang.String parentElementName)
    throws XMLValidationException;
```

Throws an XMLValidationException to indicate that a #PCDATA element was missing.

- **systemID**: The system ID of the XML data of the element
- **lineNr**: The line number in the XML data of the element
- **parentElementName**: The name of the parent element
parseDTD

```java
void parseDTD(java.lang.String publicID,
               IXMLReader reader,
               IXMLEntityResolver resolver,
               boolean external)
    throws java.lang.Exception;
```

Parses the DTD. The validator object is responsible for reading the full DTD.

- **publicID**: The public ID, which may be `null`.
- **reader**: The reader to read the DTD from.
- **resolver**: The entity resolver.
- **external**: `true` if the DTD is external

PCDataAdded

```java
void PCDataAdded(java.lang.String systemID,
                int lineNr)
    throws java.lang.Exception;
```

Indicates that a new #PCDATA element has been encountered.

- **systemID**: The system ID of the XML data of the element
- **lineNr**: The line number in the XML data of the element

setDelegate

```java
void setDelegate(IXMLValidator delegate);
```

Sets the delegate.

- **delegate**: The delegate
**setParameterEntityResolver**

```java
void setParameterEntityResolver(
    IXMLEntityResolver resolver);
```

Sets the parameter entity resolver.

resolver The resolver

**unexpectedAttribute**

```java
void unexpectedAttribute(java.lang.String systemID,
    int lineNr,
    java.lang.String elementName,
    java.lang.String attributeName)
    throws XMLValidationException;
```

Throws an XMLValidationException to indicate that an attribute is unexpected.

systemID The system ID of the XML data of the element
lineNr The line number in the XML data of the element
elementName The name of the element
attributeName The name of the unexpected attribute

**unexpectedElement**

```java
void unexpectedElement(
    java.lang.String systemID,
    int lineNr,
    java.lang.String parentElementName,
    java.lang.String unexpectedElementName)
    throws XMLValidationException;
```

Throws an XMLValidationException to indicate that an element is unexpected.

systemID The system ID of the XML data of the element
lineNr The line number in the XML data of the element
null

net.n3.nanoxml.ValidatorPlugin

parentElementName
unexpectedElementName

unexpectedPCData

void unexpectedPCData(java.lang.String systemID,
int lineNr,
java.lang.String parentElementName)
throws XMLValidationException;

Throws an XMLValidationException to indicate that a #PCDATA element was unexpected.

systemID
lineNr
parentElementName

validationError

void validationError(java.lang.String systemID,
int lineNr,
java.lang.String message,
java.lang.String elementName,
java.lang.String attributeName,
java.lang.String attributeValue)
throws XMLValidationException;

Throws a generic XMLValidationException.

systemID
lineNr
message
elementName
attributeName
attributeValue
**net.n3.nanoxml.XMLElement**

```java
<<interface>>
IXMLElement
```

**addChild**

```java
void addChild(IXMLElement child);
```

Adds a child element.

- **child**: The child to add.

**createElement**

```java
void createElement(java.lang.String fullName);
```

Creates a new empty element that is compatible with the receiver.

- **fullName**: The full name of the element.

```java
void createElement(java.lang.String fullName,
java.lang.String systemID,
int lineNr);
```

Creates a new empty element that is compatible with the receiver.

- **fullName**: The full name of the element.
- **systemID**: The system ID of the XML data where the element starts.
- **lineNr**: The line in the XML data where the element starts.
void createElement(java.lang.String fullName,
java.lang.String namespace);

Creates a new empty element that is compatible with the receiver.

fullName The full name of the element.
namespace The namespace URI

void createElement(java.lang.String fullName,
java.lang.String namespace
java.lang.String systemID,
int lineNr);

Creates a new empty element that is compatible with the receiver.

fullName The full name of the element.
namespace The namespace URI
systemID The system ID of the XML data where the element starts
lineNr The line in the XML data where the element starts

createPCDataElement
IXMLElement createPCDataElement();

Creates an element that can be used for #PCDATA content.

enumerateAttributeNames
java.util.Enumeration enumerateAttributeNames();

Returns an enumeration of all attribute names.

enumerateChildren
java.util.Enumeration enumerateChildren();

Returns an enumeration of all child elements.
equals

boolean equals(java.lang.Object obj);

Returns true if the element is equal to obj. If obj cannot be casted to an IXMLElement, the method returns false.

obj The object.

equalsXMLElement

boolean equalsXMLElement(IXMLElement elt);

Returns true if the element is equal to elt. This method is faster than the more generic equals method.

elt The other element.

getAttribute

java.lang.String getAttribute(java.lang.String name, java.lang.String defaultValue); int getAttribute(java.lang.String name, int defaultValue);

Returns the value of an attribute. If the attribute is not present, defaultValue is returned.

name The name of the attribute.
defaultValue The default value of the attribute.
**getAttribute**

```java
java.lang.String getAttribute(java.lang.String name,
                               java.lang.String namespace,
                               java.lang.String defaultValue);
```

Returns the value of an attribute. The attribute is specified using its short name and the associated namespace URI. If the attribute is not present, `defaultValue` is returned.

- **name** The name of the attribute.
- **namespace** The namespace URI, or `null` if no namespace is associated.
- **defaultValue** The default value of the attribute.

**getAttributeCount**

```java
int getAttributeCount();
```

Returns the number of attributes.

**getAttributeNamespace**

```java
java.lang.String getAttributeNamespace(
    java.lang.String name);
```

Returns the namespace URI associated with an attribute. You have to specify the full name of the attribute (i.e. the attribute with its prefix). If no namespace URI is associated with the attribute, `null` is returned.

- **name** The full name of the attribute.

**getAttributes**

```java
java.util.Properties getAttributes();
```

Returns a `Properties` object containing all the attributes of the element. The attributes will be specified using their full name.
getAttributesInNamespace

    java.util.Properties getAttributesInNamespace(
        java.lang.String namespace);

Returns a Properties object containing all the attributes which are associated with
namespace of the element. The attributes will be specified using their short names.

    namespace The namespace URI.

getAttributeType

    java.lang.String getAttributeType(
        java.lang.String name);

Returns the type of the attribute.

    name The full name of the attribute.

    java.lang.String getAttributeType(
        java.lang.String name,
        java.lang.String namespace);

Returns the type of the attribute.

    name The short name of the attribute.
    namespace The namespace URI

getChildAtIndex

    IXMLElement getChildAtIndex(int index);

Returns the child element located at a certain index.

    index The index of the child, where the first child has
        index 0.
**net.n3.nanoxml.XMLElement**

getChildren

```java
java.util.Vector getChildren();
```

Returns a vector containing all the child elements. You should not modify the vector.

getChildrenCount

```java
int getChildrenCount();
```

Returns the number of children.

getChildrenNamed

```java
java.util.Vector getChildrenNamed(java.lang.String name);
```

Returns a vector of all child elements named `name`.

```java
java.util.Vector getChildrenNamed(
    java.lang.String name,
    java.lang.String namespace);
```

Returns a vector of all child elements named `name`.

```java
name The full name of the children to search.
```

```java
name The short name of the children to search.
```

```java
namespace The namespace URI.
```

getContent

```java
java.lang.String getContent();
```

Returns the the #PCDATA content of the element. If the element has a combination of #PCDATA content and child elements, the #PCDATA sections can be retrieved as unnamed child objects. In this case, this method returns null.
**getFirstChildNamed**

    IXMLElement getFirstChildNamed(java.lang.String name);

Returns the first child named *name*. If there is no such child, null is returned.

    IXMLElement getFirstChildNamed(
            java.lang.String name,
            java.lang.String namespace);

Returns the first child named *name*. If there is no such child, null is returned.

- **name**
  The full name of the child to search.

- **namespace**
  The namespace URI.

**getFullName**

    java.lang.String getFullName();

Returns the full name of the element, which is the name including the namespace prefix. If the element only contains #PCDATA, this method return null.

**getLineNr**

    int getLineNr();

Returns the line number in the data where the element started.

**getName**

    java.lang.String getName();

Returns the name of the element, or null if the element only contains #PCDATA.
getNamespace

   java.lang.String getNamespace();

Returns the namespace URI of the element, or null if the element only contains #PCDATA or has no namespace associated with it.

getParent

   IXMLElement getParent();

Returns the parent element. If the element is the root element, the method returns null.

getSystemID

   java.lang.String getSystemID();

Returns the system ID of the data where the element started.

hasAttribute

   boolean hasAttribute(java.lang.String name);

Returns true if the element has an attribute named name.

   boolean hasAttribute(java.lang.String name, java.lang.String namespace);

Returns true if the element has an attribute named name associated with the namespace URI namespace.
**hasChildren**

    boolean hasChildren();

Returns true if the element has children.

**isLeaf**

    boolean isLeaf();

Returns true if the element has no children.

**removeAttribute**

    void removeAttribute(java.lang.String name);

Removes a attribute.

    void removeAttribute(java.lang.String name,
                         java.lang.String namespace);

Removes a attribute.

    name The full name of the attribute.
    namespace The namespace URI.

**removeChild**

    void removeChild(IXMLElement child);

Removes a child element.

    child The child to add.
**removeChildAtIndex**

```java
void removeChildAtIndex(int index);
```

Removes the child element located at a certain index.

- **index**
  The index of the child, where the first child has index 0.

**setAttribute**

```java
void setAttribute(java.lang.String name,
                  java.lang.String value);
```

Sets the value of an attribute.

- **name**
  The full name of the attribute.
- **value**
  The value of the attribute.

```java
void setAttribute(java.lang.String name,
                  java.lang.String namespace,
                  java.lang.String value);
```

Sets the value of an attribute.

- **name**
  The name of the attribute.
- **namespace**
  The namespace URI.
- **value**
  The value of the attribute.

**setContent**

```java
void setContent(java.lang.String content);
```

Sets the `#PCDATA` content. It is an error to call this method with a non-null value if there are child objects.

- **content**
  The (possible null) content.
setName

void setName(java.lang.String name);

Sets the name of the element.

name The name of the element.

void setName(java.lang.String name, java.lang.String namespace);

Sets the name of the element.

name The full name of the element.
namespace The namespace URI.

XMLElement (constructor)

XMLElement();

Creates an element that can be used for #PCDATA content.

XMLElement(java.lang.String name);

Creates an empty element.

name The name of the element.

XMLElement(java.lang.String name, java.lang.String systemID, int lineNr);

Creates an empty element.

name The name of the element.
systemID The system ID of the XML data where the element starts
lineNr The line in the XML data where the element starts
**net.n3.nanoxml.XMLEntityResolver**

```
XMLElement(java.lang.String name,
           java.lang.String namespace);
```

Creates an empty element.

- **name**: The full name of the element.
- **namespace**: The namespace URI.

```
XMLElement(java.lang.String name,
           java.lang.String namespace,
           java.lang.String systemID,
           int lineNr);
```

Creates an empty element.

- **name**: The name of the element.
- **namespace**: The namespace URI.
- **systemID**: The system ID of the XML data where the element starts.
- **lineNr**: The line in the XML data where the element starts.

**net.n3.nanoxml.XMLEntityResolver**

```
<<interface>>
IXMLEntityResolver
```

An XMLEntityResolver resolves entities. More information about custom entity resolvers can be found on page 21.
addExternalEntity

```java
void addExternalEntity(java.lang.String name,
                      java.lang.String publicID,
                      java.lang.String systemID);
```

Adds an external entity.

- **name**: The name of the entity.
- **publicID**: The public ID of the entity.
- **systemID**: The system ID of the entity.

addInternalEntity

```java
void addInternalEntity(java.lang.String name,
                       java.lang.String value);
```

Adds an internal entity.

- **name**: The name of the entity.
- **value**: The value of the entity.

getEntity

```java
java.io.Reader getEntity(IXMLReader xmlReader,
                         java.lang.String name);
```

Returns a Java reader containing the value of an entity. If the entity could not be resolved, null is returned. The method may throw an XMLParseException if necessary.

- **xmlReader**: The current NanoXML reader.
- **name**: The name of the entity.
openExternalEntity

```java
java.io.Reader openExternalEntity(
    IXMLReader xmlReader,
    java.lang.String publicID,
    java.lang.String systemID);
```

Opens an external entity. The method may throw an XMLParseException if necessary.

- xmlReader: The current NanoXML reader.
- publicID: The public ID of the entity.
- systemID: The system ID of the entity

---

net.n3.nanoxml.XMLException

An XMLException is thrown when an exception occurred while processing the XML data.

getException

```java
java.lang.Exception getException();
```

Returns the encapsulated exception, or null if no exception is encapsulated.
**getLineNr**

    int getLineNr();

Returns the line number in the XML data where the exception occurred. If there is no line number known, -1 is returned.

**getSystemID**

    java.lang.String getSystemID();

Returns the system ID of the XML data where the exception occurred. If there is no system ID known, null is returned.

**XMLException (constructor)**

    XMLException(java.lang.String msg);

Creates an exception.

    XMLException(java.lang.Exception exception);

Creates an exception.

    XMLException(java.lang.String systemID,
                  int lineNr,
                  java.lang.String msg);

Creates an exception.

**msg** The message of the exception

**exception** The encapsulated exception

**systemID** The system ID of the XML data where the exception occurred

**lineNr** The line number in the XML data where the exception occurred

**msg** The message of the exception
**net.n3.nanoxml.XMLException**

```java
XMLException(java.lang.String systemID,
            int lineNr,
            java.lang.Exception exception);
```

Creates an exception.

- **systemID** The system ID of the XML data where the exception occurred
- **lineNr** The line number in the XML data where the exception occurred
- **exception** The encapsulated exception

```java
XMLException(java.lang.String systemID,
            int lineNr,
            java.lang.Exception exception,
            java.lang.String msg,
            boolean reportParams);
```

Creates an exception.

- **systemID** The system ID of the XML data where the exception occurred
- **lineNr** The line number in the XML data where the exception occurred
- **exception** The encapsulated exception
- **msg** The message of the exception
- **reportParams** true if the the `systemID`, `lineNr` and `exception` parameters need to be appended to the message
An XMLParseException is thrown when the XML passed to the XML parser is not well-formed.

**XMLParseException (constructor)**

```java
XMLParseException(java.lang.String msg);
```

Creates an exception.

```java
msg The message of the exception
```

```java
XMLParseException(java.lang.String systemID,
        int lineNr,
        java.lang.String msg);
```

Creates an exception.

```java
systemID The system ID of the XML data where the exception occurred
lineNr The line number in the XML data where the exception occurred
msg The message of the exception
```
net.n3.nanoxml.XMLParserFactory

Creates an XML parser.

**createDefaultParser (static)**

```java
static IXMLParser createDefaultParser()
throws java.lang.ClassNotFoundException,
java.langInstantiationException,
java.langIllegalAccessException
```

Creates a default parser. The actual class is dependent on the system property `net.n3.nanoxml.XMLParser`. If this property has not been defined, the default class `net.n3.nanoxml.StdXMLParser` is used.

```java
static IXMLParser createDefaultParser(IXMLBuilder builder)
throws java.lang.ClassNotFoundException,
java.langInstantiationException,
java.langIllegalAccessException
```

Creates a default parser with a custom builder.

- **builder** The custom builder

**createParser (static)**

```java
static IXMLParser createParser(java.lang.String className,
IXMLBuilder      builder)
throws java.lang.ClassNotFoundException,
java.langInstantiationException,
java.langIllegalAccessException
```

Creates a parser with a custom builder.

- **className** The name of the class of the parser.
- **builder** The custom builder
An XMLValidationException is thrown when the XML passed to the XML parser is well-formed but not valid. There are 8 types of validation errors, each identified by a constant:

MISSING_ELEMENT:
   An element was missing

UNEXPECTED_ELEMENT:
   An unexpected element was encountered

MISSING_ATTRIBUTE:
   An attribute was missing

UNEXPECTED_ATTRIBUTE:
   An unexpected attribute was encountered

ATTRIBUTE_WITH_INVALID_VALUE:
   An attribute has an invalid value

MISSING_PCDATA:
   A #PCDATA element was missing

UNEXPECTED_PCDATA:
   An unexpected #PCDATA element was encountered

MISC_ERROR:
   Another error than those specified above was encountered
getAttributeName

java.lang.String getAttributeName();

Returns the name of the attribute in which the validation is violated. If there is no current attribute, null is returned.

getAttributeValue

java.lang.String getAttributeValue();

Returns the value of the attribute in which the validation is violated. If there is no current attribute, null is returned.

defElementName

java.lang.String getElementName();

Returns the name of the element in which the validation is violated. If there is no current element, null is returned.

XMLValidationException (constructor)

XMLValidationException(int errorType,
    java.lang.String systemID,
    int lineNr,
    java.lang.String elementName,
    java.lang.String attributeName,
    java.lang.String attributeValue,
    java.lang.String msg);

Creates a new exception.

errorType The type of validity error.
systemID The system ID from where the data came
lineNr The line number in the XML data where the exception occurred
elementName The name of the offending element (null if n/a)
attributeName The name of the offending attribute (null if n/a)
net.n3.nanoxml.XMLWriter

An XMLWriter writes XML data to a stream.

**write**

```java
void write(IXMLElement xml)
throws java.io.IOException;
void write(IXMLElement xml,
           boolean prettyPrint);
throws java.io.IOException
void write(IXMLElement xml,
           boolean prettyPrint,
           int indent)
throws java.io.IOException;
```

Writes an XML element.

- **xml**: The XML element to write
- **prettyPrint**: true if spaces need to be inserted to make the output more readable (default: false)
- **indent**: How many spaces to indent the element (default: 4; ignored if prettyPrint is false)
XMLWriter (constructor)

XMLWriter(java.io.Writer output);
XMLWriter(java.io.OutputStream output);

Creates a new XML writer.

output Where to write the output to
Index

A
addAttribute (method) 14, 27, 32, 60
addChild (method) 9, 36, 78
addExternalEntity (method) 47, 90
addInternalEntity (method) 47, 90
addPCData (method) 15, 27, 32, 60
atEOF (method) 50, 66
atEOFOfCurrentStream (method) 50, 66
attributeAdded (method) 20, 53, 56, 71
attributeWithInvalidValue (method) 21

C
createDefaultParser (method) 95
createDefaultXMLParser (method) 6, 16, 22, 23, 28
createElement (method) 9, 11, 36, 78
createParser (method) 95
createPCDataElement (method) 9, 37, 79

D
DOM 2

E
elementAttributesProcessed
   (method) 14, 27, 33, 53, 57, 61, 72
elementStarted (method) 54, 57, 58, 72
defElement (method) 14, 26, 33, 61
everateAttributeNames (method) 7, 37, 79
enumerateChildren (method) 8, 37, 79
equals (method) 38, 80

F
fileReader (method) 6, 16, 22, 23, 28, 67

G
getAttribute (method) 8, 11, 38, 80
getAttributeCount (method) 39, 81
getAttributeName (method) 97
gerAttributeNamespace (method) 39, 81
gerAttributes (method) 39, 81
gerAttributesInNamespace
   (method) 40, 82
getAttributeType (method) 40, 82
getAttributeValue (method) 97
getBuilder (method) 48, 64
getChildAtIndex (method) 40, 82
getChildren (method) 41, 83
getChildrenCount (method) 41, 83
getChildrenNamed (method) 41, 83
getChildren (method) 41, 83
getChildren (method) 41, 83
getContent (method) 8, 41, 83
getDelegate (method) 72
getElementName (method) 97
getEncoding (method) 67
getEntity (method) 21, 22, 47, 90
getException (method) 91
getFirstChildNamed (method) 42, 84
getFullName (method) 7, 10, 42, 84
getLineNr (method) 42, 51, 67, 84, 92
getName (method) 10, 42, 84
getNamespace (method) 11, 43, 85
getParameterEntityResolver (method) 55, 58, 73
getParent (method) 43, 85
getPublicID (method) 51, 67
getReader (method) 48, 64
getResolver (method) 48, 64
getResult (method) 15, 28, 34, 62
getSystemID (method) 43, 51, 67, 85, 92
getValidator (method) 21, 23, 48, 64

H
hasAttribute (method) 43, 85
hasChildren (method) 44, 86

I
invalidAttributeValue (method) 73
isLeaf (method) 44, 86
IXMLBuilder 13, 25, 28, 31, 49, 59, 65
IXMLElement 6, 8, 13, 18, 22, 23, 36, 59
IXMLEntityResolver 21, 46, 49, 56, 59, 65, 76
IXMLParser 5, 16, 20, 22, 23, 28, 48, 64
IXMLReader 6, 16, 22, 23, 28, 49, 65, 66
IXMLValidator 21, 23, 50, 56, 66, 72

M
missingAttribute (method) 73
missingElement (method) 74
missingPCData (method) 74

N
newProcessingInstruction (method) 14, 26, 34, 62
NonValidator 56

O
openExternalEntity (method) 21, 91
openStream (method) 19, 51, 68

P
parse (method) 6, 16, 22, 23, 28, 49, 65
parseDTD (method) 55, 58, 75
PCDATA 8, 9, 15, 32, 41, 55, 59, 60, 75, 83
PCDataAdded (method) 55, 59, 75
Processing Instruction 15

R
read (method) 51, 68
removeAttribute (method) 44, 86
removeChild (method) 44, 86
removeChildAtIndex (method) 45, 87

S
SAX 2, 3
setAttribute (method) 8, 11, 45, 87
setBuilder (method) 16, 28, 49, 65
setContent (method) 9, 45, 87
setDelegate (method) 21, 75
setName (method) 46, 88
setParameterEntityResolver (method) 22, 23, 56, 59, 76
setPublicID (method) 52, 68
setReader (method) 6, 16, 22, 23, 28, 49, 65
setResolver (method) 21, 22, 49, 65
setSystemID (method) 52, 68
setValidator (method) 21, 50, 66
SGML 1
startBuilding (method) 14, 26, 34, 62
startElement (method) 14, 26, 35, 63
startNewStream (method) 52, 69
StdXMLBuilder 6, 59
StdXMLBuilder (constructor) 63
StdXMLParser 6, 64
StdXMLReader 6, 16, 19, 22, 23, 28, 66
StdXMLReader (constructor) 69
stream2reader (method) 70
Streaming 13
stringReader (method) 70

U
unexpectedAttribute (method) 76
unexpectedElement (method) 76
unexpectedPCData (method) 77
unread (method) 52, 70

V
validationError (method) 77
ValidatorPlugin 20, 71

W
write (method) 6, 22, 23, 98

X
XML 1
XMLElement 8, 78
XMLElement (constructor) 11, 88
XMLEntityResolver 21, 22, 89
XMLException 91
XMLException (constructor) 92
XMLParseException 94
XMLParserFactory 5, 16, 22, 23, 28, 95
XMLValidationException 96
XMLValidationException (constructor) 97
XMLWriter 6, 22, 23, 98
XMLWriter (constructor) 99