XGA –
XML GRAMMAR API
MANUAL
XGA - XML Grammar API is the link between XML editing operations and the underlying DTD/Schema file. It provides simple methods of extracting static information about various used XML elements and their attributes and also basic operations which help in editing XML elements, such as finding valid neighbours and required attributes for example. These basic operations can be combined to provide more complex operations for dealing with today’s issues. The greatest achievement lies in an efficient algorithm to deal with the content model of a particular XML element.

Table of Contents

1. Installation
2. How to write an ElementWrapper
3. How to use XMLGrammar and ElementInfo class
1. Installation

The API comes in a JAR package and can be used as an external library as any common Java package. Source files are included in the src folder and can be easily imported by various Java IDEs for altering or expanding functionality. The current version relies on some external libraries such as NekoDTD and Xerces. Documentation for this API lies in the doc folder, but for easier understanding of the API, please refer to this manual.

File list of package content
root folder – compiled Java library
Src – source files, below a picture of the source files in Eclipse
Lib – NekoDTD, Xerces, XMLUtil, from mother project Delta
doc – javadoc generated documentation in html format, open index.html for viewing
2. How to write an ElementWrapper

The ElementWrapper class is the only class that has to be implemented by the user and is a link to the XML editing software and XGA. It provides XGA with required information of the XML elements context (next sibling, parent ...). To use XGA right from the start, a simple ElementWrapper based on the DOM interfaces from W3C is included in the package and should work with all DOM XML APIs (tested on the implementation from Xerces 2.0 API). The implementation of an individual ElementWrapper is quite straightforward and most XML APIs already own the functionality of certain methods in ElementWrapper, so that you just have to adapt your methods.

Below follows a simple demonstration of implementing an ElementWrapper with methods of the W3C interfaces and Xerces for easier understanding of the class methods return values. Malformed return values can lead to unknown program execution and fatal errors.

- ) Constructor XXXElementWrapper (… element)

The basic idea of ElementWrapper is giving XGA a connection to the XML document without any restriction on the used XML API. Therefore it is necessary to store the element object in a class variable element. This approach gives an easy way to switch between XML editor and XGA.

//CODE
public DOMElementWrapper(org.w3c.dom.Element element) {
    this.element = element;
}

- ) getName():

The getName() method simply returns the name of the element.

//CODE
public String getName(){
    return element.getTagName();
}

- ) getElement():

ElementWrapper stores the XML element in a variable and should be retrieved by this method for editing in the XML editor for example. In this implementation the DOM Element class is used to represent an element in the XML document.

//CODE
public Element getElement(){
    return element;
}
getChildren():

It is important that this method returns a list of ElementWrapper objects represented in the same order as elements declared in the XML documents (usually every getChildren method in an XML API provides that functionality).

```java
public List getChildren(){
    List elements = new ArrayList();
    NodeList nl = element.getChildNodes();
    //delete all nodes which are not of type ELEMENT
    Node currentNode;
    for(int i=0;i<nl.getLength();i++){
        currentNode = nl.item(i);
        if(currentNode.getNodeType() == Node.ELEMENT_NODE){
            elements.add(new DOMElementWrapper((Element)currentNode));
        }
    }
    return elements;
}
```

getChildrenNames():

Same as the getChildren() method, returns the children elements names as a list of String.

```java
public List getChildrenNames(){
    ...
}
```

getParent():

Returns the parent element as ElementWrapper object.

```java
public ElementWrapper get_parent(){
    Node parent = this.element.getParentNode();
    while (parent != null) {
        if (parent.getNodeType() == Node.ELEMENT_NODE)
            return new DOMElementWrapper((Element)parent);
    parent = parent.getNextSibling();
```
- getPreviousSibling():

Returns the previous sibling element as ElementWrapper object

//CODE
public ElementWrapper getPreviousSibling(){

    Node sibling = this.element.getPreviousSibling();
    while (sibling != null) {
        if (sibling.getNodeType() == Node.ELEMENT_NODE)
            return new DOMElementWrapper((Element)sibling);
        sibling = sibling.getPreviousSibling();
    }
    return null;
}

- getNextSibling():

Returns the next sibling element as ElementWrapper object

//CODE
public ElementWrapper getNextSibling(){

    Node sibling = element.getNextSibling();
    while (sibling != null) {
        if (sibling.getNodeType() == Node.ELEMENT_NODE)
            return new DOMElementWrapper((Element)sibling);
        sibling = sibling.getNextSibling();
    }
    return null;
}

- getAttributes():

public HashMap getAttributes(){

    HashMap attrs = new HashMap();
    NamedNodeMap nnm = element.getAttributes();
    int length = nnm.getLength();

    for (int i=0; i<length; i++){

        Node item = nnm.item(i);

        if(nnm.item(i).getNodeType() == Element.ATTRIBUTE_NODE)
            attrs.put(item.getNodeName(), item.getNodeValue());
    }
}
return attrs; }

- getAttributeValue(String name):

    public String getAttributeValue(String name);

- getIdentifiers():

    This method returns a list of attribute names as String objects, which are used as an ID in the XML document. The implementation of this method can be very tricky and cost intensive if not provided by the XML API such as the Xerces does with the method in DocumentImpl.getIdentifiers().

    //CODE
    public List getIdentifiers(){
        List ret = new ArrayList();
        DocumentImpl doc = (DocumentImpl)element.getOwnerDocument();
        Enumeration enu = doc.getIdentifiers();

        Object enuElem;
        while(enu.hasMoreElements()){
            enuElem = enu.nextElement();
            ret.add(enuElem);
        }
        return ret;
    }

The next method is related to the mother project Delta and uses one of its many tools to implement it easily. A more generic approach would be going to the first element in the row of elements and counting its passing siblings. The mainframe of this API will work well without this method, if you decide to do so, please make sure to remove the class DTDHelper and DTDHelperXGA as well, which provide more complex functionality based on ElementInfo and this method.

- getIndexinRow():

    Returns the elements index

    public int getIndexinRow(){
        return XMLUtils.getIndexOfChildNode(element);
    }
3. Using XMLGrammar and ElementInfo

XMLGrammar

XMLGrammar is the internal representation of the DTD/Schema and stores all the static information about elements and their attributes in a Hashmap. The interface is implemented only for DTD support so far and a detailed description of the methods in DTDGrammar follows.

- ) public void load(String filename):

Usually the load method is called by the constructor when initializing DTDGrammar, but it is free to change the underlying DTD to a XML file by calling this method at any time. This method extracts information from the DTD file, specified by the parameter filename, and fills the Hashmap with ElementDeclaration objects for each element declared in the DTD.

- ) public ElementInfo getElementInfo (ElementWrapper element):

The getElmentInfo method is the primary method to get an ElementInfo object to a specific XML element. This approach guarantees a more consistent view between ElementInfo and its linked DTDGrammar.

Please refer to the API documentation for descriptions of the remaining methods in XMLGrammar.

An example of loading a DTD with filename “test.dtd” and creating an Elementinfo to the XML element “bar” (using DOM Element)

```
DTDGrammar grammar = new DTDGrammar("test.dtd");
Element bar = … returned by an XML Parser
ElementWrapper bar = new DOMElementWrapper(bar);
Elementinfo barInfo = grammar.getElementInfo(bar);
```

ElementInfo

The ElementInfo class is the primary interface for comparing information taken from the DTD with elements of a XML document. It provides the basic functionality to find neighbours for an XML element, missing attributes, a simple validating algorithm …

ElementInfo is not instantiated directly, it needs access to an XMLGrammar object to get static information from the underlying DTD and it is linked to an ElementWrapper object that represents the actual element in the XML Document.
Please refer to the documentation for detailed descriptions to the operations provided by ElementInfo.

Example of using getAllowedNextSiblings()
Checks what siblings taken from the information stored in DTDGrammar can be appended to the current element.

Abstract of “test.dtd” DTD, element root
...
<!ELEMENT root (bar, foo?, abc)>
...

//load DTD
DTDGrammar grammar = new DTDGrammar(“test.dtd”);

//create ElementWrapper
Element bar = … returned by an XML Parser
ElementWrapper bar = new DOMElementWrapper(bar);

//get ElementInfo to bar
Elementinfo barInfo = grammar.getElementInfo(bar);

//call getAllowedNextSiblings()
List allowedNextSiblings = barInfo.getAllowedNextSiblings();

//allowedNextSiblings contains element names that can be appended to bar
//allowedNextSiblings content: [foo, abc]

//lets append abc as next element
//method getParentElement returns the first Node which is type of element
Element root = abc.getParentElement();

//create new Element abc with help of Document
Document doc = bar.getOwnerDocument();

//append abc to bar
Element abc = doc.createElement(“abc”);
root.appendChild(abc);