Parsing XML files with NanoXML

This Java ME code example describes how to parse XML files with NanoXML, one of the easiest XML libraries to use.

Overview

01 Nov 2009

The XML (eXtensible Markup Language) is a very flexible standard for describing tree-based data format. It is playing an important role due to its way of achieving interoperability among different applications by sharing structured data and being able to define new elements by its users.

In the Java ME platform, there are some APIs for parsing XML file texts. Below we can see some of the available APIs for MIDP:

- ASXMLP 020308
- kXML 2.0 alpha
- kXML 1.2
- MinML 1.7
- NanoXML 1.6.4
- TinyXML 0.7 (disabled broken link to www.gibaradunn.srac.org/tiny/)
- Xparse-J 1.1

Although there are many solutions for parsing XML file format in Java ME, NanoXML seams to be the easiest one (in my opinion, despite the lack of a good documentation!)

Source code

The best way of understanding how NanoXML works is though an example. Consider the following XML file:

```
<BreakFast_Menu>
   <food Type="Mass">
      <name>Macaroni</name>
      <price>3.0</price>
      <description>Fetucinni</description>
      <calories>213</calories>
   </food>
   <food Type="desert">
      <name>chocolat</name>
      <price>6.0</price>
      <description>laka</description>
      <calories>334</calories>
   </food>
</BreakFast_Menu>
```

The example above describes a menu containing two types of food (mass and a type of desert). Through NanoXML API, I will show you how to create and read an XML file with this kind of information. The code below demonstrates how to create the XML file. For each new food, we create a new child and adds it to the tree structure (mainElement):

```
//Creates a node in the XML tree
private kXMLElement createElement(String name, String str) {
 kXMLElement element = new kXMLElement(name); //adds the name of the node
 element.setContent(str); //adds the content of the current node
 return element;
}

//Adds an element in the XML tree
public void addNode(String foodName, double price, ...) {
 kXMLElement child = new kXMLElement("food"); //creates a node

        //addChild method adds a node's child
 child.addChild(createElement("name", foodName));
 child.addChild(createElement("price", String.valueOf(price)));
 child.addChild(createElement("description", description));
 child.addChild(createElement("calories", String.valueOf(calories)));

 child.addProperty(type, typeName); //adds a property to the node
 mainElement.addChild(child); //adds the node to the main XML tree
}
...

addNode("Macaroni", ...);
addNode("Chocolat", ...);
```

To save this new data, we will use JSR 75 (FileConnection).

```
private void initFileSystem() {
    connection = (FileConnection)Connector.open("file:///"+
                  rootFileSystem+"Foods/");
    if(!connection.exists())
 connection.mkdir(); //Creates a new folder, if does not exist!

    connection = (FileConnection) Connector.open("file:///"+
                  rootFileSystem+"Foods/food.xml", Connector.READ_WRITE);
    if(!connection.exists())
       connection.create(); //Creates a new file, if does not exist!
}

private void save() {
    try {
```


(C) Copyright Nokia 2013. All rights reserved.

```
 OutputStream out = connection.openOutputStream();
 out.write(mainElement.toString().getBytes());
 out.flush();
 out.close();
    } catch (IOException e) {
 e.printStackTrace();
    }
}
```

To retrieve the information just saved, we just open an input stream reader and apply the data to a kXMLElement reference, as follows:

```
//the parseFromReader methods adds the file contents into the mainElement reference as an XML-based text.
private readFile() {
     mainElement.parseFromReader(new InputStreamReader(
                                        connection.openInputStream()));
}

//Reads the XML file recursively
public void readMenu(kXMLElement element) {
     Enumeration e = element.enumerateChildren();

     kXMLElement childElement = null;
     while(e.hasMoreElements()) {
 childElement = (kXMLElement) e.nextElement();
 if(childElement.getTagName().equals("food")) {
            formView.append(new StringItem("Tipo",
                                   childElement.getProperty("Tipo")));
     readMenu(childElement);
 } else {
     formView.append(new StringItem(
            childElement.getTagName(), childElement.getContents()));
 }
}
}
```

getTagName() and getContents() methods return the tag name ("name", "price", "calories") and it's values, respectively. On the other hand, childElement.getProperty(String) returns an attribute tag value, desert or mass, for instance.

Another example aims at creating an XML file through an XML-based text. Consider the following XML file:

```
<person>
   <name>Thiago</name>
   <age>23</age>
</person>
```

Once getting a text formatted in XML, to retrieving an XML-based text and each node using NanoXML is as follows:

```
public class Example2 extends MIDlet {

    private Display d;
    private Form f;

    public Example2() {
 d = Display.getDisplay(this);
 f = new Form("Person");

 kXMLElement element = new kXMLElement();
        // This text could be retrieved from the Web
 element.parseString("<person><name>Thiago</name><age>23</age></person>");


 Enumeration elements = element.enumerateChildren(); //gets the element's childreen
 while(elements.hasMoreElements()) {
  kXMLElement e = ((kXMLElement)elements.nextElement());
  String tag = e.getTagName(); //tag's name: "name" and "age"
  f.append(new StringItem(tag, e.getContents())); //e.getContents() returns the content of the node ("thiago", 23)
 }

    }

    protected void destroyApp(boolean arg0) throws MIDletStateChangeException {}
    protected void pauseApp() {}

    protected void startApp() throws MIDletStateChangeException {
 this.d.setCurrent(this.f);
    }

}
```

The result of this example is:

(C) Copyright Nokia 2013. All rights reserved.

Download

The examples described here is available in: File:ExampleNanoXML.zip

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.