Parsing XML in Java ME

Version 0.4, Draft

INFO

COPYRIGHT

About This Document

This document gives a short introduction to XML and explains on how to parse xml file in Java ME. This document does not explain XML in details. Developers are requested to know XML before reading this document. This document describes about KXML Parser and how to parse XML using KXML parser. Sample code snippet demonstrating usage of KXML Parser is provided at the end of this document.

Scope

This document is intended for MIDP developers wishing to develop mobile applications that use XML. It assumes good knowledge of Java programming language and XML. This document focuses on XML parsing and therefore explaining the Java technology and XML is out of the scope in this documentation.

To know more about Java ME basics and Java programming language, refer to the Knowledge Base under Samsung Mobile Innovator (SMI).

http://innovator.samsungmobile.com/platform.main.do?platformId=3

Table of Contents

Introduction

There are many server client applications that mainly deal with the processing of data. Processing of data involves communication of server and client applications in which transfer of data takes place in a manner both client as well server can understand. But how do we represent the data so both can understand.

In simple terms data representation is done in a structured way that can easily be understandable to human and machine. XML is one such way of representing structured data. Let's consider the following example: we have to send book order information from server to client. This data can be transmitted to the client in a way like shown below in Listing 1:

Listing 1: Sample data format

```
Parsing XML in Java ME
Article that explains how to parse XML in Java ME
SMI
1234
KB
Java ME
```

A more simple and understandable way to represent the same data in XML is shown below in Listing 2:

Listing 2: XML data format

```
<Article>Parsing XML in Java ME</Article>
<Description>Article that explains how to parse XML in Java ME</Description>
<Author>SMI</Author>
<ArticleId>1234</ArticleId>
<Content>KB</Content>
<Platform>Java ME</Platform>
```

Obviously the second example gives a clear understanding of the data. Its advantages can be summed up as:

1. Simple easy to understand.
2. Robust. Here each data is marked up with what the data means. SMI represents the Author name, KB represents the Content etc
3. Extensibility. You can very well add more information as and when required. XML can very well handle unexceptional changes or addition. For example if the order of data as changed let's say Content has moved up immediately after Article. In this case also you need not to worry about reading data since each data is marked.
4. Ease of use.

Fortunately you don't need to do the hard work of writing the code to get the data. You need to use XML parser which reads the information from XML and returns you the required data.

XML Terminology

XML or XML Documents are text. Each XML document is a sequence of characters. These characters are taken from the Unicode character set. An XML document is a tree. It has a root node that contains various child nodes. Some of these child nodes have children of their own. Others are leaf nodes that have no children.

There are roughly five different kinds of nodes in an XML tree:

Root

Also known as the document node, this is the abstract node that contains the entire XML document. Its children include comments, processing instructions, and the root element of the document.

Element (Tags, Attributes)

An XML element with a name, a list of attributes, a list of in-scope namespaces, and a list of children.

Text (data)

The parsed character data between two tags (or any other kind of non-text node).

Comment

An XML comment such as <!-- This needs to be fixed. -->. The contents of the comment are its data. A comment does not have any children.

Processing instruction

A processing instruction such as <?xml-stylesheet type="text/css" href="order.css"?> A processing instruction has a target and a value. It does not have any children.

XML Parser

XML Parsing falls into two categories. Which one to choose depends upon what kind of XML documents the application uses and the memory used by the application.

1. Tree Model Parser

A. Document Object Model:

Document Object Model (DOM) parser is a tree model parser. DOM parser reads through the entire document, builds the entire XML document representation in memory and then hands the calling program the whole chunk of memory. DOM parser occupies extensive memory.

2. Stream Based Parser (Event based)

A. Push Parser:

A Push Parser reads through the document and as the parser encounters elements in an XML, it notifies the application through callback methods (listener objects). SAX parser is one such example of push parser.

B. Pull Parser:

A Pull Parser is opposite of push parser. Parser provides data only when the application requests it. The application drives the parser through the document by repeatedly requesting the next piece.

Which parser to use in application depends on the characteristics of the application and XML documents.

kXML Parser

kXML is a small XML pull parser, specially designed for constrained environments such as Applets, Personal Java or MIDP devices. The latest version of kXML parser is kXML2. kXML3.0 will soon be released. kXML parser is available at http://kxml.sourceforge.net/

Pull based XML parsing combines some of the advantages of SAX and DOM:

- In contrast to push parsers (SAX), pull parsers such as kXML make it possible to model the XML processing routines after the structure of the processed XML document. Events processing is similar to an InputStream. If a part of the stream requires special handling, the parser can simply be delegated to a specialized method by handing over the parser.

- While the above is also possible with an explicit DOM, DOM usually requires that the whole document structure is present in main memory.

- In contrast to DOM based parsing, the XML events are accessible immediately when they are available, it is not necessary to wait for the whole tree to build up.

kXML Version History

1. **kXML1**

   kXML1 is a simple pull parser, based on event objects.

   kXML1 is archived at kxml.objectweb.org and can be downloaded from http://kxml.objectweb.org/software/downloads/. kXML1 is now deprecated, please use kXML2 instead.

2. **kXML2**

   This is the current version of kXML. In contrast to kXML1, it features cursor API instead of event objects, leading to a reduced footprint and less object creation

overhead. kXML2 is released under the BSD license. kXML2 can be downloaded from http://sourceforge.net/projects/kxml/files/kxml2/

3. **kXML3**

   kXML3 will split the parser and API support available in versions for both, XmlPull and StAX. kXML3 will be released in future.

kXML1

kXML1 supports XML pull parsing support, XML namespace support, XML writing support and optional DOM, WAP support. Following are the packages of kXML1. The important classes required for xml parsing are highlighted.

Table 1: kXML1 packages

Packages	Interface	Classes	Exception
org.kxml	XmlIO	**Attribute**, PrefixMap, **Xml**.	
org.kxml.io		AbstractXmlWriter, State, XmlWriter	ParseException
org.kxml.kdom		Document, Element, Node, TreeParser	
org.kxml.parser		AbstractXmlParser, **ParseEvent**, StartTag,Tag, **XmlParser**	
org.kxml.wap		WapExtensionEvent , Wbxml, WbxmlParser, WbxmlWriter, Wml, WmlParser, WmlParser	

Working with kXML1

Let's start with XML parsing. We need to have the following requirements for parsing xml file.

Requirements

1. First thing we need to do is to download the zip / jar file of respective version (the one you will be using). This document explains parsing using kXML 1.21 version.
2. Second put this downloaded zip / jar file in the "lib" folder of your Samsung SDK Project.

XML Parsing

Let's try parsing the SMI Java Knowledge Base RSS Feed. The output xml data of this feed would be something similar to the data shown in Listing 3.

Listing 3: RSS Feed XML data

```
<?xml version="1.0" encoding="utf-8"?>
<rss version="2.0"
        xmlns:dc="http://purl.org/dc/elements/1.1/"
        xmlns:sy="http://purl.org/rss/1.0/modules/syndication/"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
        <channel>
           <item>
             <title>Screen cast : Introduction of Samsung Java ME SDK</title>
             <description> This video tutorial will cover in brief about Samsung Java ME
                                SDK and Java ME building environment</description>
               <category>Knowledge Base</category>
             </item>
        </channel>
</rss>
```

Listing 4 shows the code flow parsing RSS Feed XML. Following is the summary of Listing 4. Please have a look at the kXML documentation before proceeding further.

Listing 4: XML parsing object creation

```
byte[] xmlByteArray = xmlStr.getBytes();
ByteArrayInputStream xmlStream = new ByteArrayInputStream( xmlByteArray );
InputStreamReader xmlReader = new
InputStreamReader( xmlStream );
XmlParser parser = new XmlParser( xmlReader );

        OR

HttpConnection hc = (HttpConnection)Connector.open(url);
InputStream is = hc.openInputStream();
Reader reader = new InputStreamReader(is);
XmlParser parser = new XmlParser(reader);
```

1. We need to create XMLParser object. XMLParser constructor takes java.io.Reader as parameter. Since xml parser is a stream based parser, we need to provide stream of data. This can be done in either ways.
2. The RSS data is made available to parse XML in the form of a String. XML parser works on a stream of bytes. For this purpose, String is converted into a byte array, which is used to construct an instance of ByteArrayInputStream. The ByteArrayInputStream in turn creates an instance of InputStreamReader, which creates an instance of an XMLParser
3. OR, Application can open a URL connection and get RSS data on an InputStream. Then InputStream is made available to the XMLParser through an InputStreamReader. The ByteArrayInputStream in turn creates an instance of InputStreamReader, which creates an instance of an XMLParser.
4. Application creates an instance of XMLParser to walk its way through the document. On every item tag it finds, the parser looks for the sub tags title, description, category in order to find the text that you want to retrieve. This process goes on recursively. This process is listed in Listing 5.
5. After instance creation, parser calls read() method. ParseEvent object is returned whenever the read() finds the next available event.
6. This object contains valuable information, such as the event type (whether it represents the start of a tag, the end of a tag, the end of a document, text, or whitespace), the event name (that is, the tag name), and the event text (that is, the text enclosed between the start and end tags).

7. The ParseEvent generates the Event types Xml.START_TAG, Xml.END_TAG, Xml.END_DOCUMENT, Xml.WHITESPACE, and Xml.TEXT when it encounters the start of a tag, end of a tag, end of a document, whitespace, and text between tags, respectively.
8. Here we are interested in item, title, description tags and related information. Listing 5 shows the parsing code.

Listing 5: rss xml parsing code

```
public void parse(XmlParser parser) throws Exception {
        boolean xmlParsingDone = false;
        boolean error = false;
        String title = "";
        String desc = "";
        String errorstr = "";
        String errortype = "";
        while (!xmlParsingDone) {
            ParseEvent event = parser.read();
            ParseEvent pe;
            switch (event.getType()) {
                // For example, <title>
                case Xml.START_TAG:
                    // see API doc of StartTag for more access methods
                    // Pick up Title for display
                    String tagName = event.getName();
                    if ("title".equals(tagName)) {
                        pe = parser.read();
                        title = pe.getText();
                        menuForm.append("\nTitle:\n" + title);
                    }
                    // Pick up description for display
                    if ("description".equals(tagName)) {
                        pe = parser.read();
                        desc = pe.getText();
                        menuForm.append("\nDescription:\n" + desc);
                    }
```

```
                // Pick up error for display
                if ("error".equals(tagName)) {
                    pe = parser.read();
                    errortype = pe.getText();
                    menuForm.append("\nError type:\n" + errortype);
                    error = true;
                }
                // Pick up error for display
                if ("message".equals(tagName)) {
                    pe = parser.read();
                    errorstr = pe.getText();
                    menuForm.append("\nError:\n" + errorstr);
                    xmlParsingDone = true;
                }
            break;
            // For example </title?
            case Xml.END_TAG:
            break;
            // For example </rss>
            case Xml.END_DOCUMENT:
                xmlParsingDone = true;
            break;
            // For example, the text between tags
            case Xml.TEXT:
            break;
            case Xml.WHITESPACE:
            break;
            default:
        }
    }
}
```

kXML2

kXML1 support has now been deprecated. So we should now use kXML2 for parsing xml files. In contrast to kXML1, it features cursor API instead of event objects, leading to a reduced footprint and less object creation overhead. In addition it supports SyncML, Wml, Wv support.

kXML2 implements the XmlPull API. For more information please visit http://www.xmlpull.org/.

Following are the packages of kXM2. The important classes required for xml parsing are highlighted.

Table 2: kXML2 packages

Packages	Interface	Classes	Exception
org.kxml2.io		**KXmlParser**, KXmlSerializer	
org.kxml2.kdom		Document, Element, Node	
org.kxml2.wap	Wbxml	WbxmlParser, WbxmlSerializer	
org.kxml2.wap.syncml		SyncML	
org.kxml2.wap.wml		Wml	
org.kxml2.wap.wv		WV	
org.xmlpull.v1	**XmlPullParser**, XmlSerializer	XmlPullParserFactory	XmlPullParserException

Working with kXML2

Requirements remain same as discussed in kXML1. We are using kXML 2.3.0 version.

General XML content can be parsed with the XML pull API using a loop advancing to the next event and a switch statement that depends on the event type. However, when using XML for data transfer (in contrast to text documents), most XML elements contain either only text or only other elements (possibly with further sub-elements).

For those common cases, the parsing process can be simplified significantly by using the XmlPull API methods *nextTag()* and *nextText*.() Additionally, the method *require()* may optionally be used to assert a certain parser state.

Requirements

1. First thing we need to do is to download the jar file of respective version (the one you will be using). This document explains parsing using kXML 2.3.0 version.
2. Second put this downloaded jar file in the "lib" folder of your Samsung SDK Project.

XML Parsing

Let's try parsing the same SMI Java Knowledge Base RSS Feed using kXML2.3.0 xml parser. XML data remains same as shown in Listing 3. The code flow to parse the xml data is shown in Listing 7.

1. We need to create an instance of KXMLParser. The steps upto 4 are almost same except here the inputstream is passed through setInput method as shown in Listing 6.

Listing 6: Creating KXMLparser object

```
HttpConnection hc = (HttpConnection) Connector.open(url);
KXmlParser parser = new KXmlParser();
InputStream rssStream = hc.openInputStream();
InputStreamReader isr = new InputStreamReader(rssStream);
parser.setInput(isr);
```

2. Please have a look at the kXML2 documentation and XmlPull v1 API before proceeding further. kXML2 documentation can be found at http://kxml.sourceforge.net/kxml2/javadoc/ and XmlPull v1 API can be found at http://www.xmlpull.org/v1/doc/api/org/xmlpull/v1/package-summary.html
3. After instance creation, parser calls require() method. require() methods internally advances to the required tag event type and tag name.
4. nextTag() advances to the next start or end tag, skipping insignificant events such as white space, comments and PIs.
5. nextText() returns the text content of the corresponding element. nextText() requires that the current position is a start tag.
6. Here we are interested in item, title, description tags and related information. Listing 7 shows the parsing code.

Listing 7: rss xml parsing code

```
public void parse(KXmlParser parser) throws Exception {
        boolean error = false;
        parser.nextTag();
        parser.require(XmlPullParser.START_TAG, null, "rss");
        parser.nextTag();
        parser.require(XmlPullParser.START_TAG, null, "channel");
        parser.nextTag();
        while (!xmlParsingDone) {
            int eventType = parser.getEventType();
            if (parser.getEventType() != XmlPullParser.END_TAG) {
                String nodeName = parser.getName();
                if (nodeName.compareTo("item") == 0) {
                    parser.nextTag();
                    while(parser.getEventType() != XmlPullParser.END_TAG) {
                        String tagName = parser.getName();
                        if (tagName.compareTo("title") == 0) {
                            String title = parser.nextText();
                            menuForm.append("\nTitle:\n" + title);
                        } else if (tagName.compareTo("description") == 0) {
                            String description = parser.nextText();
```

```
                    menuForm.append("\nDescription:\n" + description);
                } else if (tagName.compareTo("link") == 0) {
                    String link = parser.nextText();
                } else if (tagName.compareTo("error") == 0) {
                    String errorType = parser.nextText();
                    menuForm.append("\nError type:\n" + errorType);
                    error = true;
                } else if (tagName.compareTo("message") == 0) {
                    String errorMessage = parser.nextText();
                    menuForm.append("\nError:\n" + errorMessage);
                    xmlParsingDone = true;
                } else {
                    parser.skipSubTree();
                }
                parser.nextTag();
            }
        } else {
            parser.skipSubTree();
        }
        parser.nextTag();
    }
  }
}
```

Sample Example

kXML1

Class: RSSMidlet

```
import javax.microedition.midlet.*;
import org.kxml.Xml;
import org.kxml.parser.ParseEvent;
import org.kxml.parser.XmlParser;

public class RSSMidlet extends MIDlet implements CommandListener,Runnable {
    private List menuList;
    private Display display;
    private Form menuForm;
    private boolean xmlParsingDone = false;

    private Command selectCommand = new Command("Select", Command.OK, 3);
    private Command backCommand = new Command("Back", Command.BACK, 1);
    private Command exitCommand = new Command("Exit", Command.EXIT, 2);

    private final String[] menus = {
        "Device Specs",
        "Tools & SDKs",
        "Knowledge Base",
        "Discussion Boards",
        "Innov Lab",
        "FAQs",};

    private final String[] url = {
        "http://innovator.samsungmobile.com/servlet/rss/products.xml?platformId=3",
        "http://innovator.samsungmobile.com/servlet/rss/downloads.xml?platformId=3",
        "http://innovator.samsungmobile.com/servlet/rss/"+
        "samsung_s60_symbian_best.xml?platformId=3",
        "http://innovator.samsungmobile.com/servlet/rss/discussion.xml?platformId=3",
```

```
    "http://innovator.samsungmobile.com/servlet/rss/innov.xml?platformId=2",
    "http://innovator.samsungmobile.com/servlet/rss/faq.xml?platformId=3",};

public void startApp() {
    display = Display.getDisplay(this);
    menuForm = new Form("");
    menuForm.addCommand(backCommand);
    menuForm.setCommandListener(this);
    menuList = new List("SMI JAVA RSS MENU", List.IMPLICIT, menus, null);
    menuList.setSelectCommand(selectCommand);
    menuList.addCommand(exitCommand);
    menuList.setCommandListener(this);
    display.setCurrent(menuList);
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional) {
    notifyDestroyed();
}

public void commandAction(Command c, Displayable d) {
    if (c == exitCommand) {
        destroyApp(true);
    } else if (c == selectCommand) {
        new Thread(this).start();
    } else if (c == backCommand) {
        xmlParsingDone = true;
        display.setCurrent(menuList);
    }
}

public void run()
{
```

```
        menuForm.deleteAll();
        display.setCurrent(menuForm);
        int index = menuList.getSelectedIndex();
        parseXML(url[index]);
    }

    private void parseXML(String url) {
        try {
            menuForm.append("Parsing XML");
            HttpConnection hc = (HttpConnection) Connector.open(url);
            int responseCode = hc.getResponseCode();
            menuForm.append("\nResponse code is "+responseCode);
            if( responseCode == HttpConnection.HTTP_OK)
            {
                InputStream is = hc.openInputStream();
                Reader reader = new InputStreamReader(is);
                XmlParser parser = new XmlParser(reader);
                parse(parser);
                is.close();
            }else
            {
                StringItem str = new StringItem("\nError: Unable to connect", "\nResponse
                                                        code is "+responseCode);
                menuForm.append(str);
            }
            hc.close();
        } catch (IOException ex) {
            ex.printStackTrace();
            menuForm.append("\nException caught at HttpConnectin "+ex.getMessage());
        } catch (Exception ex) {
            ex.printStackTrace();
            menuForm.append("\nException caught at parsing xml "+ex.getMessage());
        }
    }
```

```
public void parse(XmlParser parser) throws Exception {
    xmlParsingDone = false;
    boolean error = false;
    String title = "";
    String description = "";
    String errorMessage = "";
    String errorType = "";
    while (!xmlParsingDone) {
        ParseEvent event = parser.read();
        ParseEvent pe;
        switch (event.getType()) {
            case Xml.START_TAG:
                String tagName = event.getName();
                if ("title".equals(tagName)) {
                    pe = parser.read();
                    title = pe.getText();
                    menuForm.append("\nTitle:\n" + title);
                }
                if ("description".equals(tagName)) {
                    pe = parser.read();
                    description = pe.getText();
                    menuForm.append("\nDescription:\n" + description);
                }
                if ("error".equals(tagName)) {
                    pe = parser.read();
                    errorType = pe.getText();
                    menuForm.append("\nError type:\n" + errorType);
                    error = true;
                }
                if ("message".equals(tagName)) {
                    pe = parser.read();
                    errorMessage = pe.getText();
                    menuForm.append("\nError:\n" + errorMessage);
                    xmlParsingDone = true;
                }  break;
```

```
                case Xml.END_TAG:
                    tagName = event.getName();
                    break;
                case Xml.END_DOCUMENT:
                    xmlParsingDone = true;
                    break;
                case Xml.TEXT:
                    break;
                case Xml.WHITESPACE:
                    break;
                default:
            }
        }
    }
}
```

kXML2

Class: RSSMidlet

```
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.Reader;
import javax.microedition.io.Connector;
import javax.microedition.io.HttpConnection;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.List;
import javax.microedition.lcdui.StringItem;
import javax.microedition.midlet.*;
import org.kxml2.io.KXmlParser;
import org.xmlpull.v1.XmlPullParser;

public class RSSMidlet extends MIDlet implements CommandListener, Runnable {
    private List menuList;
    private Display display;
    private Form menuForm;
    private boolean xmlParsingDone = false;
    private Command selectCommand = new Command("Select", Command.OK, 3);
    private Command backCommand = new Command("Back", Command.BACK, 1);
    private Command exitCommand = new Command("Exit", Command.EXIT, 2);
    private final String[] menus = {
        "Device Specs",
        "Tools & SDKs",
        "Knowledge Base",
        "Discussion Boards",
        "Innov Lab",
```

```
    "FAQs",};

private final String[] url = {
    "http://innovator.samsungmobile.com/servlet/rss/products.xml?platformId=3",
    "http://innovator.samsungmobile.com/servlet/rss/downloads.xml?platformId=3",
    "http://innovator.samsungmobile.com/servlet/rss/"+
    "samsung_s60_symbian_best.xml?platformId=3",
    "http://innovator.samsungmobile.com/servlet/rss/discussion.xml?platformId=3",
    "http://innovator.samsungmobile.com/servlet/rss/innov.xml?platformId=2",
    "http://innovator.samsungmobile.com/servlet/rss/faq.xml?platformId=3",};

public void startApp() {
    display = Display.getDisplay(this);
    menuForm = new Form("");
    menuForm.addCommand(backCommand);
    menuForm.setCommandListener(this);
    menuList = new List("SMI JAVA RSS MENU", List.IMPLICIT, menus, null);
    menuList.setSelectCommand(selectCommand);
    menuList.addCommand(exitCommand);
    menuList.setCommandListener(this);
    display.setCurrent(menuList);
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional) {
    notifyDestroyed();
}

public void commandAction(Command c, Displayable d) {
    if (c == exitCommand) {
        destroyApp(true);
    } else if (c == selectCommand) {
        new Thread(this).start();
```

```
        } else if (c == backCommand) {
            xmlParsingDone = true;
            display.setCurrent(menuList);
        }
    }

    public void run() {
        menuForm.deleteAll();
        display.setCurrent(menuForm);
        int index = menuList.getSelectedIndex();
        parseXML(url[index]);
    }

    private void parseXML(String url) {
        try {
            xmlParsingDone = false;
            menuForm.append("Parsing XML");
            HttpConnection hc = (HttpConnection) Connector.open(url);
            int responseCode = hc.getResponseCode();
            menuForm.append("\nResponse code is " + responseCode);
            if (responseCode == HttpConnection.HTTP_OK) {
                KXmlParser parser = new KXmlParser();
                InputStream rssStream = hc.openInputStream();
                InputStreamReader isr = new InputStreamReader(rssStream);
                parser.setInput(isr);
                parse(parser);
                isr.close();
                rssStream.close();
            } else {
                StringItem str = new StringItem("\nError: Unable to connect", "\nResponse
                                                code is " + responseCode);
                menuForm.append(str);
            }
            hc.close();
        } catch (IOException ex) {
```

```
            ex.printStackTrace();
            menuForm.append("\nException caught at HttpConnectin " +
ex.getMessage());
        } catch (Exception ex) {
            ex.printStackTrace();
            menuForm.append("\nException caught at parsing xml " + ex.getMessage());
        }
    }

    public void parse(KXmlParser parser) throws Exception {
        boolean error = false;
        parser.nextTag();
        parser.require(XmlPullParser.START_TAG, null, "rss");
        parser.nextTag();
        parser.require(XmlPullParser.START_TAG, null, "channel");
        parser.nextTag();
        while (!xmlParsingDone) {
            int eventType = parser.getEventType();
            if (parser.getEventType() != XmlPullParser.END_TAG) {
                String nodeName = parser.getName();
                if (nodeName.compareTo("item") == 0) {
                    parser.nextTag();
                    while (parser.getEventType() != XmlPullParser.END_TAG) {
                        String tagName = parser.getName();
                        if (tagName.compareTo("title") == 0) {
                            String title = parser.nextText();
                            menuForm.append("\nTitle:\n" + title);
                        } else if (tagName.compareTo("description") == 0) {
                            String description = parser.nextText();
                            menuForm.append("\nDescription:\n" + description);
                        } else if (tagName.compareTo("link") == 0) {
                            String link = parser.nextText();
                        } else if (tagName.compareTo("error") == 0) {
                            String errorType = parser.nextText();
                            menuForm.append("\nError type:\n" + errorType);
```

```
                    error = true;
                } else if (tagName.compareTo("message") == 0) {
                    String errorMessage = parser.nextText();
                    menuForm.append("\nError:\n" + errorMessage);
                    xmlParsingDone = true;
                } else {
                    parser.skipSubTree();
                }
                parser.nextTag();
            }
        } else {
            parser.skipSubTree();
        }
        parser.nextTag();
    }
  }
}
```