Lesson 14

Processing XML and JSON Encoded Data

Victor Matos
Cleveland State University

Processing XML and JSON Encoded Data

Why Internet data is encoded?

- Two important data issues that need to be considered by developers attempting to securely exchange data on the web are: **size** and **transparency**.
- **Size** is clearly important because small data packets require less transmission time than equivalent larger versions of the same data.
- **Transparency** gives data a human-readable quality that allows developers to better understand the nature of their data an create portable cross-platform solutions more rapidly.
- In this lesson we will discuss **XML** and **JSON**, two public-domain strategies commonly used for encoding data to be exchanged on a cross-platform environment.

Reading XML Data

What is XML data ?

What is XML?

- **Extensible Markup Language** (**XML**) is a set of rules established by the W3C organization. The rules provide a framework for uniformly encoding documents in a *human readable* form.
- XML is similar to HTML but all the **<tags>** are user-defined.

Example: Defining a golf Tee-Time

```
<?xml version='1.0' encoding='UTF-8'?>
<time golfcourse="Augusta Ntl" tournament="The Masters" >
  <hour> 21 </hour>
  <minutes> 25 </minutes>
  <seconds> 45 </seconds>
  <zone> UTC-05:00 </zone>
</time>
```

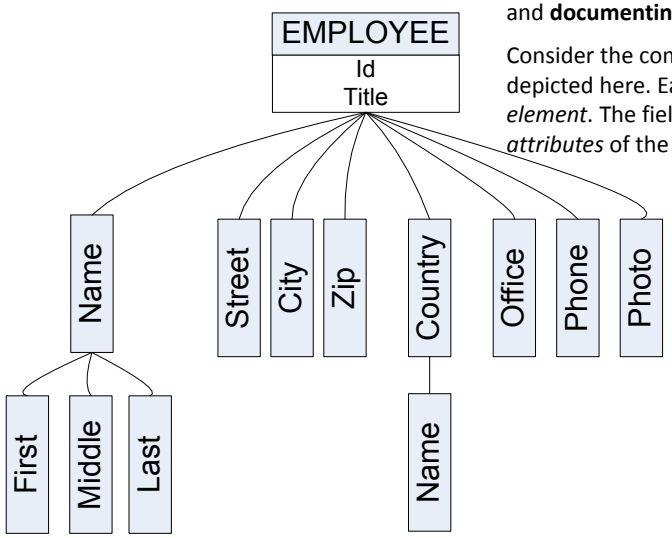

Reading XML Data

Why should XML be used ?

- The main role of XML is to facilitate a transparent exchange of data over the Internet.
- Example of technologies using XML include: RSS , Atom, SOAP, XHTML, KML, Xquery, Xpath, OpenDocument, OpenMath, etc.(see http://en.wikipedia.org/wiki/List_of_XML_markup_languages)
- Several document management productivity tools default to XML format for internal data storage. Example: Microsoft Office, OpenOffice.org, and Apple's iWork.
- Android OS relies heavily on XML to save its various resources such as layouts, string-sets, manifest, etc.

Reading XML Data

Example 1. How is XML used?

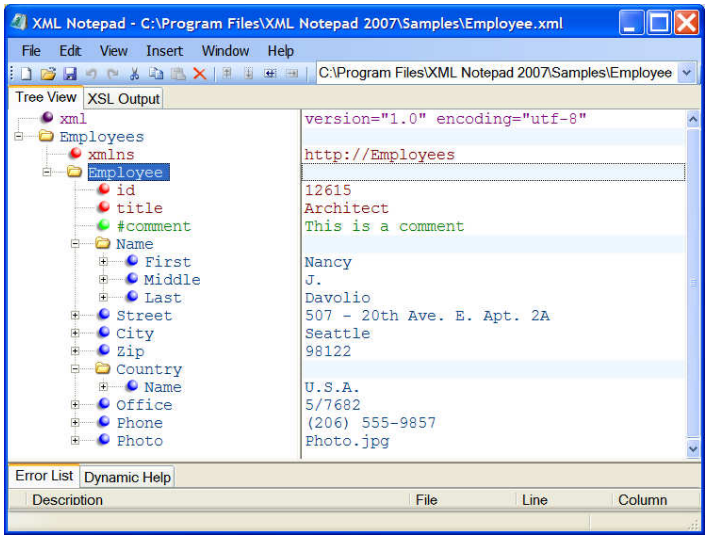


XML is used for **defining** (.xsd files) and **documenting** (.xml) classes.

Consider the complex **Employee** class depicted here. Each node is an XML *element*. The fields **Id** and **Title** are *attributes* of the **Employee** class.

Reading XML Data

Example 1. How is XML used?



This image was made using **Microsoft XML Notepad**.

On the left, the structure of the Employee class is depicted as a tree.

On the right, a data sample from the current node is provided.

Link: https://www.microsoft.com/en-us/download/details.aspx?id=7973

Reading XML Data

Example 1. How is XML used?

The XML fragment below depicts the structure and data for a set of Employee objects.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Employees xmlns="http://Employees">
    <Employee  id="12615" title="Architect">
        <!-- This is a comment  -->
        <Name>
            <First>Nancy</First>
            <Middle>J.</Middle>
            <Last>Davolio</Last>
        </Name>
        <Street>507  20th Ave. E. Apt. 2A</Street>
        <City>Seattle</City>
        <Zip>98122</Zip>
        <Country>
            <Name>U.S.A.</Name>
        </Country>
        <Office>5/7682</Office>
        <Phone>(206) 5559857</Phone>
        <Photo>Photo.jpg</Photo>
    </Employee>
    <Employee>
        ...  ...  ...
    </Employee>
</Employees>
```

Attributes: **id, title**

Element: **Street**

Example taken from:
Microsoft XmlNotepad 2007
http://www.microsoft.com/downloads/en/details.aspx?familyid=72d6aa49787d4118ba5f4f30fe913628&displaylang=en

Reading XML Data

Example 1. Employee.xsd – Schema Definition *(fragment)*

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
  attributeFormDefault="unqualified" targetNamespace="http://Employees" xmlns="http://Employees">

 <xs:complexType name="Country">
       <xs:sequence>
          <xs:element name="Name" type="xs:string" default="U.S.A." />
       </xs:sequence>
       <xs:attribute name="code" type="xs:language">
         <xs:annotation>
           <xs:documentation>Registered IANA country code – Format xxxx. Example: enus.
           </xs:documentation>
         </xs:annotation>
       </xs:attribute>
 </xs:complexType>

 <xs:simpleType name="City">
  <xs:restriction base="xs:string">
    <xs:minLength value="1" />
    <xs:maxLength value="50" />
  </xs:restriction>
 </xs:simpleType>

 <xs:simpleType name="Zip">
 <xs:restriction base="xs:positiveInteger">
  <xs:maxInclusive value="99999" />
  <xs:minInclusive value="00001" />
  </xs:restriction>
  </xs:simpleType>
```

Only a few lines shown here

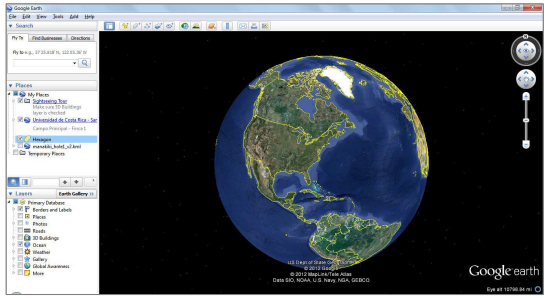
Reading XML Data

Example 2. Example: KML and Geographic Data

KeyHole Markup Languale (KML) [1] is a file format used to record geographic data. Currently it is under the purview of the Open Geospatial Consortium (**OGC**) [2].

The goal of KML is to become the *single international standard language for expressing geographic annotation and visualization on existing or future web-based online and mobile maps and earth browsers.*

Example of applications using the format are:
Google Earth,
Google Maps, and
Google Maps for Mobile Apps.

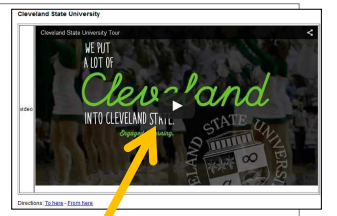


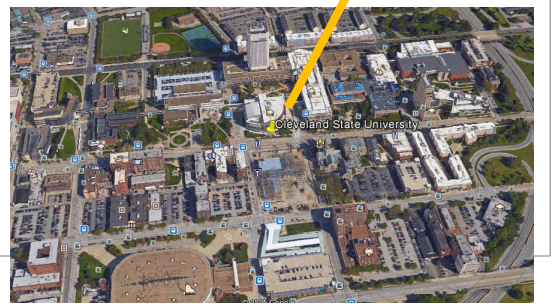
References:
[1] Displaying KML files in your Maps application. https://developers.google.com/maps/tutorials/kml/
[2] Open Geospatial Consortium. http://www.opengeospatial.org/standards/kml#overview

Reading XML Data

Example 2A. Example: KML and Geographic Data

```
<Placemark>

  <name>Cleveland State University</name>

  <ExtendedData>
    <Data name="video">
      <value><![CDATA[<iframe width="640" height="360"
        src="http://www.youtube.com/embed/es9KEhVlOiw"
        frameborder="0"
        allowfullscreen></iframe><br><br>]]></value>
    </Data>
  </ExtendedData>

  <Point>
    <coordinates  -81.675281,
                  41.501792,
                  0
    </coordinates>
  </Point>

</Placemark>
```

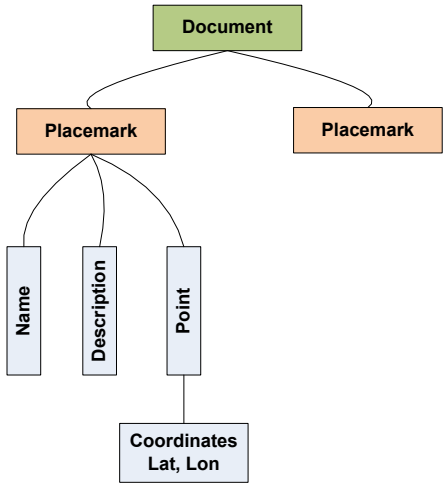




Reference: https://developers.google.com/maps/tutorials/kml/
https://developers.google.com/kml/documentation/kmlreference?hl=en

Reading XML Data

Example 2B. Example: KML and Geographic Data



In this example a Document consists of various **Placemark** elements. The markers identify a set of points-of-interest.

Each of our <placemarks> includes a
- *Name*,
- *Description,* and a
- *Geo-Point* including: latitude, longitude and altitude.

Reference: http://code.google.com/apis/kml/documentation/kml_tut.html

Reading XML Data

Example2B. Mapping with KML *(fragment)*

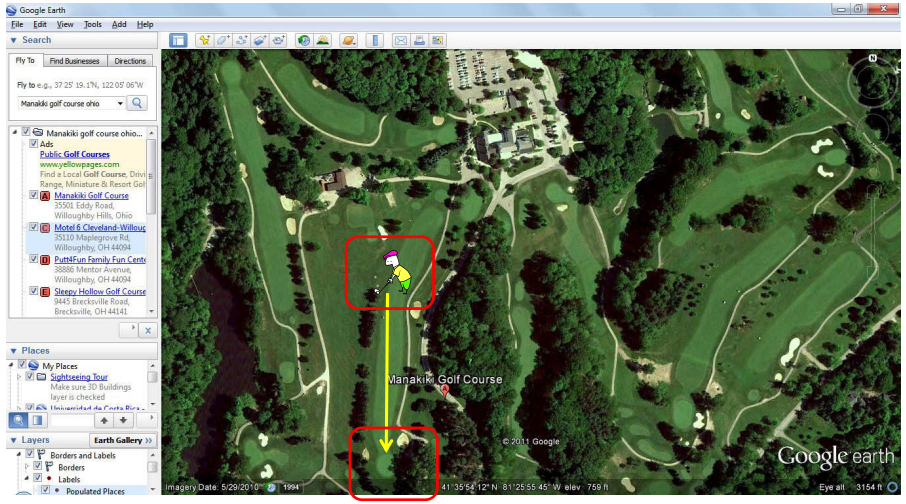
```
<?xml version="1.0" encoding="utf-8" ?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>

    <gcPlace gcName="Manakiki Golf Course" gcCity="Willoughby Hills" gcState="Ohio" />

    <Placemark>
          <name  par="4"  yards="390" >Tee  Hole 1</name>
          <Point>
           <coordinates>81.4324182271957,41.5984273639879,0</coordinates>
           </Point>
    </Placemark>

    <Placemark>
          <name>Front of Green  Hole 1</name>
          <Point>
           <coordinates>81.433182656765,41.5955730479591,0</coordinates>
           </Point>
     </Placemark>

     <Placemark>
          <name>Middle of Green  Hole 1</name>
          <Point>
           <coordinates>81.4331665635109,41.5954647298964,0</coordinates>
           </Point>
     </Placemark>

 </Document>
 </kml>
```

Reference: Data downloaded from http://www.bbgpsgolf.com/viewcourselistg.php

Reading XML Data

Example 3. Helping Golfers with KML

After a rather mediocre Tee-shot, the player on the picture is trying to reach the green. How far away is it ?, what club should he pick ?



Reference: http://code.google.com/apis/kml/documentation/kml_tut.html

Reading XML Data

Example 3. Helping Golfers with KML

Typical Distances for (*good*) Amateur Players

Club	Men	Women
Driver	200-230-260	150-175-200
3-wood	180-215-235	125-150-180
2-Hybrid	170-195-210	105-135-170
3-Hybrid	160-180-200	100-125-160
4-iron	150-170-185	90-120-150
5-iron	140-160-170	80-110-140
6-iron	130-150-160	70-100-130
7-iron	120-140-150	65-90-120
8-iron	110-130-140	60-80-110
9-iron	95-115-130	55-70-95
PW	80-105-120	50-60-80
SW	60-80-100	40-50-60

By the end of the lesson you should know how to create a golf GPS device.

"Your main problem is you are standing too close to the ball... after you have hit it."

Source: http://golfcartoons.blogspot.com/ Golf Monthly 2009

Reading XML Data

Strategies for Reading/Parsing an XML File

SAX
Simple API for XML

W3C

- Several approaches are available
- Here we will explore two options:

OPTION 1 A SAX (Simple API for XML) XmlPullParser	OPTION 2 W3C-DOM Document Builder
You traverse the document programmatically looking for the beginning and ending of element tags, their associated text and internal attributes.	A Document Builder object dissects the XML document producing an equivalent tree-like representation. Nodes in the tree are treated as familiar Java ArrayLists.
References: http://www.saxproject.org/ http://www.w3.org/DOM/	The World Wide Web Consortium (W3C.org) is an *"international community that develops open standards to ensure the long-term growth of the Web".*

Reading XML Data

SAX
Simple API for XML

Example 4. SAX-Parsing a Resource XML File

- In this example we will read a XML file saved in the app's **/res/xml** folder. The file contains a set of KML **placemark** nodes pointing to locations in a golf course (tee-boxes, front/center/back of each green, obstacles, etc)
- A **SAX** (Simple API for XML) **XmlPullParser** will traverse the document using the **.next()** method to detect the following main *eventTypes*

```
START_TAG
TEXT
END_TAG
END_DOCUMENT
```

- When the beginning of a tag is recognized, we will use the **.getName()** method to grab the tag's name.
- We will use the method **.getText()** to extract data after TEXT event.

Reference: http://www.saxproject.org/

Reading XML Data

SAX
Simple API for XML

Example 4. SAX-Parsing a Resource XML File

Inner **attributes** from an **<element>** can be extracted using the methods:

```
.getAttributeCount()
.getAttributeName()
.getAttributeValue()
```

Consider the **name**-element in the example below:

```
<name par="4" yards="390" >Tee Hole 1</name>
```

Attributes	
AttributeName	**AttributeValue**
par	4
yards	390

2 attr

Element: "name"

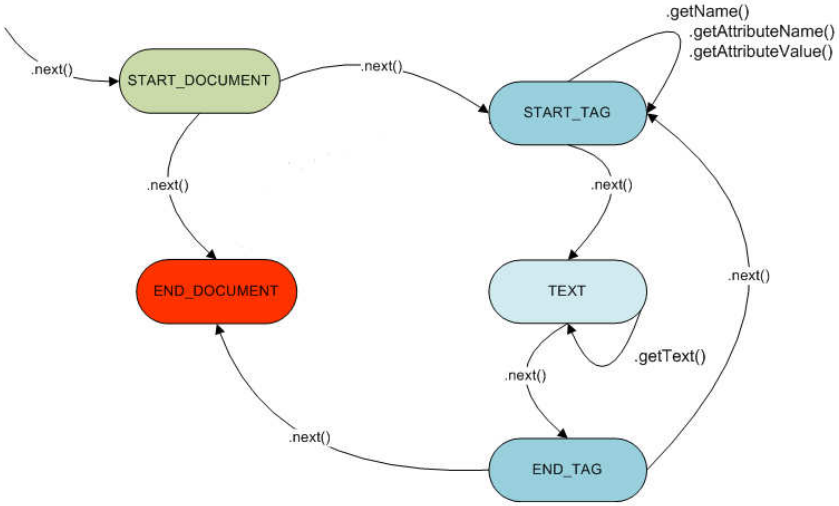
Text: "Tee Hole 1"

Reading XML Data

SAX
Simple API for XML

Example 4. SAX-Parsing of an Arbitrary XML File

Diagram showing the life-cycle of the **XmlPullParser** class. Any well-formed XML input document could be processed as suggested in the figure.



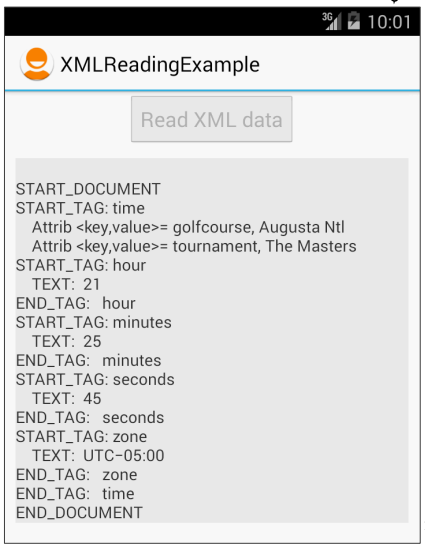
Reading XML Data

SAX
Simple API for XML

Example 4. SAX-Parsing a Resource XML File

Parsing the Tee-Time XML file listed below

```
<?xml version='1.0' encoding='UTF-8'?>
<time golfcourse="Augusta Ntl"
      tournament="The Masters" >
   <hour> 21 </hour>
   <minutes> 25 </minutes>
   <seconds> 45 </seconds>
   <zone> UTC-05:00 </zone>
</time>
```

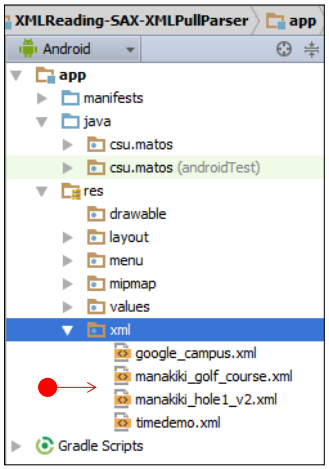


Reading XML Data

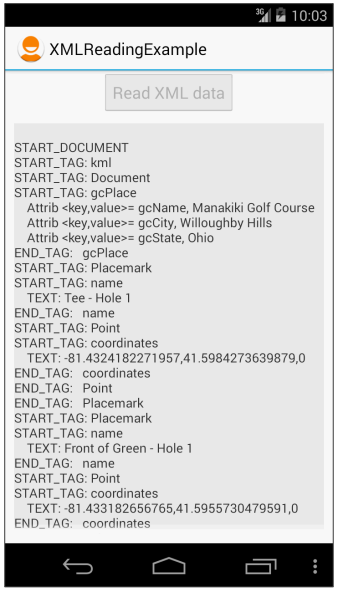
SAX
Simple API for XML

Example 4. SAX-Parsing of a Resource KML File

The XMLPullParser used in the previous example will now be used to dissect a KML file holding <placemark> elements mapping a golf course.



Observation: AT the time of writing, Android Studio 1.3 requires .kml files to be renamed with the extension .xml



Reading XML Data

SAX
Simple API for XML

Example 4. SAX-Parsing of a Resource KML File

```
<?xml version='1.0' encoding='UTF-8'?>
<kml xmlns='http://www.opengis.net/kml/2.2'>

<Document>

<gcPlace
   gcName="Manakiki Golf Course"
   gcCity="Willoughby Hills"
   gcState="Ohio"   >
</gcPlace>

  <Placemark>
    <name>Tee - Hole 1</name>
    <Point>
      <coordinates>-81.4324182271957,41.5984273639879,0</coordinates>
    </Point>
  </Placemark>

  <Placemark>
    <name>Front of Green - Hole 1</name>
    <Point>
      <coordinates>-81.433182656765,41.5955730479591,0</coordinates>
    </Point>
  </Placemark>

   . . .
  </Document>
</kml>
```

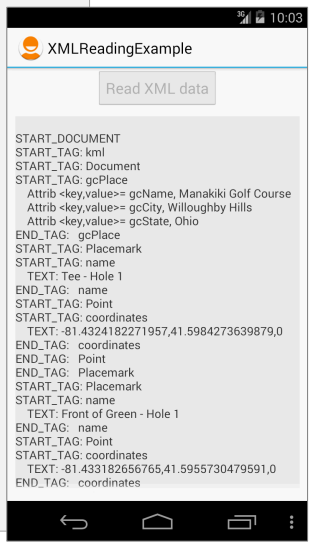
This is an abbreviated version of the geographic **KML** file read by the app

Reading XML Data

SAX
Simple API for XML

Example 4. App's Screen Layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Button
        android:id="@+id/btnReadXml"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Read XML data" />

    <ScrollView
        android:id="@+id/ScrollView01"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="2"
        android:padding="10dp">

        <TextView
            android:id="@+id/txtMsg"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:background="#ffeeeeee"  />
    </ScrollView>

</LinearLayout>
```



Reading XML Data

SAX
Simple API for XML

Example 4. SAX-Parsing of a Resource KML File

```
public class ActivityMain extends Activity {

   private TextView txtMsg;
   Button btnGoParser;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      txtMsg = (TextView) findViewById(R.id.txtMsg);
      btnGoParser = (Button) findViewById(R.id.btnReadXml);

      btnGoParser.setOnClickListener(new View.OnClickListener() {

         @Override
         public void onClick(View v) {
            btnGoParser.setEnabled(false);
            // do slow XML reading in a separated thread (AsyncTask)
            Integer xmlResFile = R.xml.manakiki_hole1_v2;

            new backgroundAsyncTask().execute(xmlResFile);
         }
      });
   }// onCreate
```


Reading XML Data

SAX
Simple API for XML

Example 4. SAX-Parsing of a Resource KML File

```
   public class  backgroundAsyncTask extends
                 AsyncTask<Integer, Void, StringBuilder> {
      ProgressDialog dialog = new ProgressDialog(ActivityMain.this);

      @Override
      protected void onPostExecute(StringBuilder result) {
         super.onPostExecute(result);
         dialog.dismiss();
         txtMsg.setText(result.toString());
      }

      @Override
      protected void onPreExecute() {
         super.onPreExecute();
         dialog.setMessage("Please wait...");
         dialog.setCancelable(false);
         dialog.show();

      }

      @Override
      protected void onProgressUpdate(Void... values) {
         super.onProgressUpdate(values);
         // Nothing here. Needed by the interface
      }
```

Reading XML Data

SAX
Simple API for XML

Example 4. SAX-Parsing of a Resource KML File

```
    @Override
    protected StringBuilder doInBackground(Integer... params) {
       int xmlResFile = params[0];
5 →    XmlPullParser parser = getResources().getXml(xmlResFile);
       StringBuilder stringBuilder = new StringBuilder();
       String nodeText = "";
       String nodeName = "";
       try {
          int eventType = -1;
6 →       while (eventType != XmlPullParser.END_DOCUMENT) {
             eventType = parser.next();

             if (eventType == XmlPullParser.START_DOCUMENT) {
                stringBuilder.append("\nSTART_DOCUMENT");

             } else if (eventType == XmlPullParser.END_DOCUMENT) {
                stringBuilder.append("\nEND_DOCUMENT");

             } else if (eventType == XmlPullParser.START_TAG) {
                nodeName = parser.getName();
                stringBuilder.append("\nSTART_TAG: " + nodeName);
7 →             stringBuilder.append(getAttributes(parser));

             } else if (eventType == XmlPullParser.END_TAG) {
                nodeName = parser.getName();
                stringBuilder.append("\nEND_TAG:   " + nodeName );
```


Reading XML Data

SAX
Simple API for XML

Example 4. SAX-Parsing of a Resource KML File

```
             } else if (eventType == XmlPullParser.TEXT) {
                nodeText = parser.getText();
                stringBuilder.append("\n    TEXT: " + nodeText);

             }
          }

       } catch (Exception e) {
          Log.e("<<PARSING ERROR>>", e.getMessage());

       }
       return stringBuilder;

    }// doInBackground
```


Reading XML Data

SAX
Simple API for XML

Example 4. SAX-Parsing of a Resource KML File

```
    private String getAttributes(XmlPullParser parser) {
       StringBuilder stringBuilder = new StringBuilder();
       // trying to detect inner attributes nested inside a node tag
       String name = parser.getName();

       if (name != null) {
          int size = parser.getAttributeCount();
          for (int i = 0; i < size; i++) {
8 →          String attrName = parser.getAttributeName(i);
             String attrValue = parser.getAttributeValue(i);
             stringBuilder.append("\n    Attrib <key,value>= "
                                  + attrName + ", " + attrValue);
          }
       }
       return stringBuilder.toString();
    }// getAttribures

 }// backgroundAsyncTask

}// ActivityMain
```


Reading XML Data

SAX
Simple API for XML

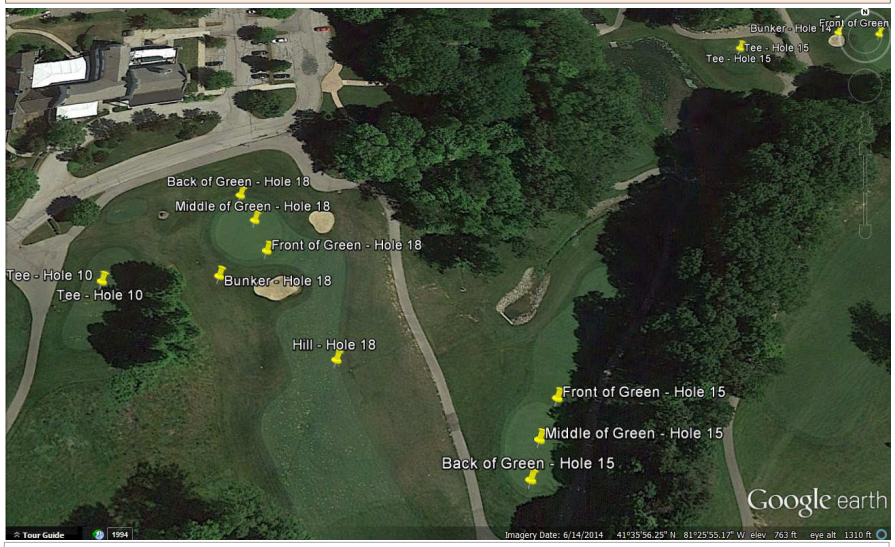
Example 4. Comments

1. The XML file is held as an internal resource in the **/res/xml** folder.
2. Invoke the reading-parsing process inside an AsyncTask. Pass the XML file id as argument to the slow background thread.
3. The parsing process has finished. The progress dialog box is dismissed.
4. Housekeeping. Create and show a simple **ProgressDialog** box so the user gets reassured about his task been taken care of.
5. Create an **XmlPullParser** using the supplied file resource.
6. The *while-loop* implements the process of stepping through the SAX's parser state diagram. Each call to .next() provides a new token. The if-then logic decides what event is in progress and from there the process continues looking for text, attributes, or end event.
7. When a START_TAG event is detected the parser checks for possible inner attributes. If found, they are reported as a sequence of <key, value> pairs.
8. The method **getAttributes()** extracts attributes (if possible). A loop driven by the count of those attributes attempts to get the name and value of each pair 'name=value' for the current element. The result is returned as a string.

Reading XML Data

SAX
Simple API for XML

Example 4. Comments



A segment of the Google Earth's map depicted using the .kml file of Example4.

Reading XML Data

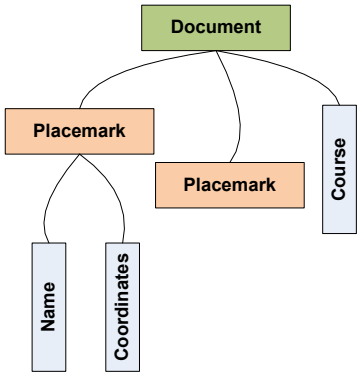
Example 5. The W3C DocumentBuilder Class

In this example we will explore a second approach for decoding an XML document.

1. A W3C **DocumentBuilder** parser will be used for decoding an arbitrary (well-formed) XML file.
2. In our example, the input file is stored externally in the **SD card**.
3. The file includes various elements: <course>, <name>, <coordinates>.
4. For each <element> -type in the document, the parser will create a **NodeList** collection to store the **text** and **attributes** held in each node type.
5. For instance, our sample XML file describes a regulation golf course. The Document contains three type of elements: <name>, <coordinates>, and <course>.
6. The <name> elements identify important locations in the course such as: 'Tee-Box Hole1', 'Front of Green – Hole1', 'Bunker1-GreenLeft-Hole1', …, 'Back of Green – Hole18'.
7. The NodeList made for the <coordinates> elements contain the latitude and longitude of each entry held in the <name> list.
8. The <course> element uses xml-attributes to keep the course's name, phone, and total length.

Reading XML Data

Example 5. The W3C DocumentBuilder Class



Parser's Strategy

<Elements> from the input XML file become nodes in an internal tree representation of the dataset. The node labeled **<Document>** acts as the root of the tree.

Your Turn

PHASE 1. For each selected XML element you request the construction of a **NodeList** collection using the method:

- `document.getElementsByTagName(...)`

PHASE2. Explore an individual node element from a NodeList using the methods:

- `list.item(i)`
- `node.getName()`
- `node.getValue()`
- `node.getFirstChild()`
- `node.getAttributes(), etc.`

Reading XML Data

Example 5. The W3C DocumentBuilder Class

Only a few entries are shown for the input XML file used in this example. Later, we will request lists to be made for the elements: **course**, **name**, and **coordinate**.

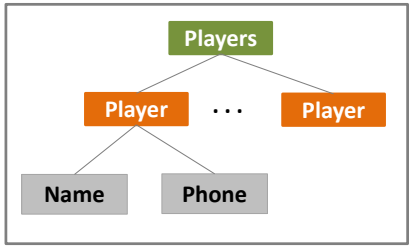
```
<?xml version="1.0" encoding="utf-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Document>
    <course  phone="(440)942-2500" length="6500">Manakiki Golf Course</course>
    <Placemark>
      <name>Tee Box - Hole 1</name>
      <Point>
        <coordinates>-81.4324182271957,41.5984273639879,0</coordinates>
      </Point>
    </Placemark>
    <Placemark>
      <name>Front of Green - Hole 1</name>
      <Point>
        <coordinates>-81.433182656765,41.5955730479591,0</coordinates>
      </Point>
    </Placemark>
    <Placemark>
      <name>Middle of Green - Hole 1</name>
      <Point>
        <coordinates>-81.4331665635109,41.5954647298964,0</coordinates>
      </Point>
    </Placemark>
```

Reading XML Data

Example 5. The W3C DocumentBuilder Class

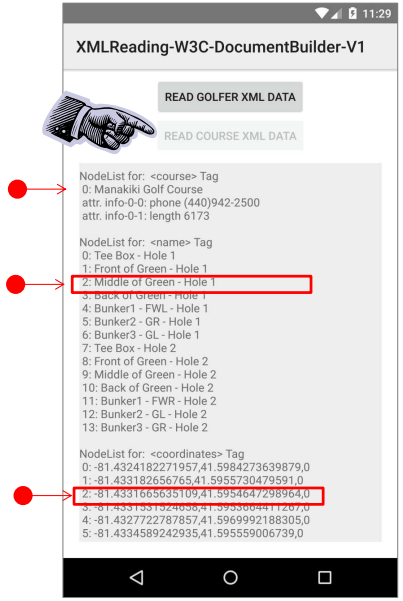
A second XML data set (Golfers.xml) is used to store the name and phone of a few friendly golfers.

```
<Players xmlns="http://Players">
  <!--  Arnie and Lee do not like early tee times  -->
  <Player>
    <Name>Arnie Palmer</Name>
    <Phone>555-0001</Phone>
  </Player>
  <Player>
    <Name>Lee Trevino</Name>
    <Phone>555-0002</Phone>
  </Player>
  <Player>
    <Name>Annika Sorenstan</Name>
    <Phone>555-0003</Phone>
  </Player>
  <Player>
    <Name>Happy Gilmore</Name>
    <Phone>555-0004</Phone>
  </Player>
  <Player>
    <Name>Ty Webb</Name>
    <Phone>555-0005</Phone>
  </Player>
</Players>
```



Reading XML Data

Example 5. The W3C DocumentBuilder Class



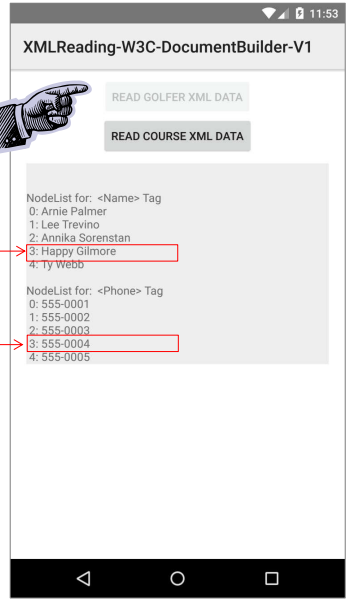
The screen shows the result of parsing the xml-geo-data file describing the golf course.

The first arrow point to the <course> element. There is only one in the XML file. We have extracted its text, and attributes (phone, length).

The second arrows points to the third node in the <name> list (say <name> [2]) which holds the value: *"Middle of the Green – Hole1"*, The last arrow point to its coordinates <coordinates>[2]

Reading XML Data

Example 5. The W3C DocumentBuilder Class



This screen shows the data obtained from parsing the "Golfers.xml" file.

For each element <name> and <phone> the parser produces a NodeList.

Observe the correspondence between the lists (parallel arrays). For instance player number 3 is Happy Gilmore, and his phone number is phone 3 which in our sample is 555-0004.

Example 5. App's Screen Layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"   android:layout_height="match_parent"
              android:orientation="vertical" >
    <Button
        android:id="@+id/btnReadXmlPlayers"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Read GOLFER XML data"/>
    <Button
        android:id="@+id/btnReadXmlCourse"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Read COURSE XML data"/>
    <ScrollView
        android:id="@+id/ScrollView01"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="2"
        android:padding="10dp">
        <TextView
            android:id="@+id/txtMsg"
            android:layout_width="match_parent"
            android:layout_height="wrap_content“ />
    </ScrollView>
</LinearLayout>
```

Reading XML Data

Example 5. The W3C DocumentBuilder Class


```
package csu.matos;

import android.app.Activity;
import android.app.ProgressDialog;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
```


Reading XML Data

Example 5. The W3C DocumentBuilder Class


```
public class MainActivity extends Activity {

   private TextView txtMsg;
   Button btnGoParsePlayers;
   Button btnGoParseCourse;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      txtMsg = (TextView) findViewById(R.id.txtMsg);

      btnGoParsePlayers = (Button) findViewById(R.id.btnReadXmlPlayers);
      btnGoParsePlayers.setOnClickListener(new View.OnClickListener() {

         @Override
         public void onClick(View v) {
            btnGoParsePlayers.setEnabled(false);
            // KML stored in the SD card - needs: READ_EXTERNAL_DEVICE
            // Example1: a group of <Player> friends stored in the "golfers.xml" file
            // holding elements: <Name>, <Phone>  Case sensitive!!!
            // ------------------------------------------------------------------------
            // do slow XML reading in a separated thread
            new BackgroundAsyncTask().execute("Golfers.xml", "Name", "Phone");

         }
      });
```


Reading XML Data

Example 5. The W3C DocumentBuilder Class


```
      btnGoParseCourse = (Button) findViewById(R.id.btnReadXmlCourse);
      btnGoParseCourse.setOnClickListener(new View.OnClickListener() {

         @Override
         public void onClick(View v) {

            btnGoParseCourse.setEnabled(false);

            // KML stored in the SD card - needs: READ_EXTERNAL_DEVICE
            // ------------------------------------------------------
            // Example2: this xml file includes elements: <course>,
            // <name>, <coordinates> for just a few holes
            // String xmlFile = "manakiki_holes1and2.xml";
            // ------------------------------------------------------
            // do slow XML reading in a separated thread
            new BackgroundAsyncTask().execute("manakiki_holes1and2.xml",
                                              "course", "name", "coordinates");

         }

      });

   }// onCreate
```


Reading XML Data

Example 5. The W3C DocumentBuilder Class


```
   private class BackgroundAsyncTask extends AsyncTask<String, Void, String> {

      ProgressDialog dialog = new ProgressDialog(MainActivity.this);

      @Override
      protected void onPostExecute(String result) {
         super.onPostExecute(result);
         dialog.dismiss();
         txtMsg.setText(result.toString());
      }

      @Override
      protected void onPreExecute() {
         super.onPreExecute();
         dialog.setMessage("Please wait...");
         dialog.setCancelable(false);
         dialog.show();
      }

      @Override
      protected void onProgressUpdate(Void... values) {
         super.onProgressUpdate(values);
      }

      @Override
      protected String doInBackground(String... params) {
         return useW3CParser(params);
      }// doInBackground

   }// backgroundAsyncTask
```

Reading XML Data

Example 5. The W3C DocumentBuilder Class


```
private String useW3CParser(String... params) {
   // params contains: xml-file-name followed by <element>s
   // for example: "Golfers.xml", "Name", "Phone"
   // CAUTION: XML is case-sensitive.

   int n = params.length;     // total number of parameters

   String xmlFileName = params[0];      // xml file name

   String[] elementName = new String[n - 1]; // element names
   for (int i = 0; i < n - 1; i++) elementName[i] = params[i + 1];

   StringBuilder str = new StringBuilder();
   try {
      String kmlFile = Environment.getExternalStorageDirectory()
                                  .getPath() + "/" + xmlFileName;

      InputStream is = new FileInputStream(kmlFile);

3 →   DocumentBuilder docBuilder = DocumentBuilderFactory.newInstance()
                                                         .newDocumentBuilder();

      Document document = docBuilder.parse(is);

      if (document == null) {
         Log.v("REALLY BAD!!!!", "document was NOT made by parser");
         return "BAD-ERROR";
      }
```


Reading XML Data

Example 5. The W3C DocumentBuilder Class


```
      // make a NodeList for each given <element> - prepare data to be shown
      NodeList[] elementList = new NodeList[n];

      for (int i = 0; i < n - 1; i++) {
         //make a collection of <elements> for each name in params[i+1]
4 →      elementList[i] = document.getElementsByTagName(elementName[i]);
         //dissect node elementList[i] looking for its enclosed attributes and text
5 →      str.append(getTextAndAttributesFromNode(elementList[i], elementName[i]));

      }

   } catch (FileNotFoundException e) {
      Log.e("W3C Error", e.getMessage());

   } catch (ParserConfigurationException e) {
      Log.e("W3C Error", e.getMessage());

   } catch (SAXException e) {
      Log.e("W3C Error", e.getMessage());

   } catch (IOException e) {
      Log.e("W3C Error", e.getMessage());

   }
   return str.toString();

}// useW3cOrgDocumentBuilder
```


Reading XML Data

Example 5. The W3C DocumentBuilder Class


```
private Object getTextAndAttributesFromNode(NodeList list, String strElementName) {

   StringBuilder str = new StringBuilder();
   // dealing with the <strElementName> tag
   str.append("\n\nNodeList for:  <" + strElementName + "> Tag");
   for (int i = 0; i < list.getLength(); i++) {

6 →   // extract TEXT enclosed inside <element> tags
      Node node = list.item(i);
      String text = node.getTextContent();
      str.append("\n " + i + ": " + text);

      // get ATTRIBUTES inside the current element
7 →   int size = node.getAttributes().getLength();
      for (int j = 0; j < size; j++) {
         String attrName = node.getAttributes().item(j).getNodeName();
         String attrValue = node.getAttributes().item(j).getNodeValue();

         str.append("\n attr. info-" + i + "-" + j + ": " + attrName
                    + " " + attrValue);
      }
   }

   return str;

}//getAllDataFromNodeList

}// ActivityMain
```


Reading XML Data

Example 5. Comments

1. Do the *slow* parsing process inside an AsyncTask thread. Pass a variable number of arguments including: the external XML file's name, followed by the name of each element to be extracted.
2. The **doInBackground** method calls **`useW3CParser`** where all the work is to be actually done.
3. The method **useW3CParser** instantiates a **DocumentBuilder** worker to accept the data stream coming from the XML file. This method creates an internal tree-like representation of the structured XML-document.
4. The tree version of the document is traversed and **NodeLists** are made for the elements: <name>, <coordinates> and <course> [Example 2].
5. Each of the lists is visited to report their corresponding contents.
6. For each node extract the text (if any) held between the beginning and end tags.
7. For each node extract its internal attribute (if any) in the form of <key, value> pairs.

Reading JSON Data

What is JSON ?

JSON (**J**ava**S**cript **O**bject **N**otation) is a plain-text formatting protocol for encoding and decoding hierarchically structured data.

1. JSON is based on JavaScript Programming Language
2. It is language and platform independent.
3. Arguably, it is easier to read and write than XML.
4. A JSON encoded data-set is based on the manipulation of two common programming constructs: simple **arrays** and **objects**.
5. Each **object** is represented as an **associative-array** holding a collection of attributes and their values.
6. An attribute's value could be a simple data-type or another nested JSON object.

Reading JSON Data

Syntaxt Rules

Object[0]	Object[1]		Object[M]

{
Attr_1 : value1
...
Attr_n0 : value_n0
}
	{
	Attr_1 : value1
	...
	Attr_n1 : value_n1
	}
			{
			Attr_1 : value1
			...
			Attr_nM : value_nM
			}

- Individual data items are represented as **key : value** pairs
- Data items are separated by **commas**
- Objects are enclosed inside **curly braces { }**
- Arrays of objects are delimited by **square brackets [ ]**

Example. A JSON array of three Person objects, each holding name & age.

```
"Person" :
[
    {"name":"Daenerys", "age":20},
    {"name":"Arya", "age":12},
    {"name":"Cersei", "age":35}
]
```


Reading JSON Data

Example 6. Using JSON & PHP

```
<?php
   // define native PHP objects
   $person0 = array('name' => 'Daenerys', 'age' => 20);
   $person1 = array('name' => 'Arya',  'age' => 12);
   $person2 = array('name' => 'Cersei',  'age' => 35);

   $people = array($person0, $person1);
   $people[2] = $person2;
   // PHP objects are converted to JSON format
   echo "<p>" . json_encode($person1);

1 → $jsondata = json_encode($people);
   echo "<p>" . $jsondata;
   // JSON formated data is decoded into native PHP objects
2 → $parr = json_decode($jsondata);

   echo "<p>" . var_export($parr);

   echo "<br>" . $parr['0']->name;
   echo "<br>" . $parr['0']->age;
   ?>
```

In this example we create a **PHP** array in which each cell is itself an associative array holding a person's name and age.

Reading JSON Data

Example 6. Using JSON & PHP

This is the output produced by the previous example:

3 →
{"name":"Arya","age":12} *A single JSON encoded Person object*

4 →
A JSON array of Person objects

[{"name":"Daenerys","age":20},{"name":"Arya","age":12},{"name":"Cersei","age":35}]

5 →
array (*Decoding from JSON to a PHP associative array*
  0 => stdClass::__set_state(array('name' => 'Daenerys', 'age' => 20,)),
  1 => stdClass::__set_state(array('name' => 'Arya', 'age' => 12,)),
  2 => stdClass::__set_state(array('name' => 'Cersei', 'age' => 35,)),)

6 →
Daenerys *Individual values of a selected PHP object*
20

Reading JSON Data

Example 6. Using JSON & PHP

Comments

1. The PHP associative array **$people** is a collection of <key,value> pairs, the statement **$jsonData = json_encode($people)** converts this representation into a JSON string [{…}, {…}, {…}]

2. The statement **json_decode($jsonData)** reverses the JSON string into an ordinary PHP associative array.

3. JSON objects are enclosed by curly-braces { "name":"Arya","age":12}

4. JSON arrays hold their comma separated items inside braces […]

5. When a JSON string representing an array of PHP objects is decoded, it becomes a PHP associative array. Each cell holds the object's attributes.

6. The expression **$parr['0']->name** allows access to the "name" attribute of the zero-th object in the PHP array.

Reading JSON Data

Example 6. Using JSON & PHP

```php
<?php
   $person0 = array('name' => 'Daenerys', 'age' => 20);
   $person1 = array('name' => 'Arya',     'age' => 12);
   $person2 = array('name' => 'Cersei',   'age' => 35);

   $people = array($person0, $person1, $person2);

   $jsondata = json_encode($people);
   echo "<p>JSON Encoded Data <br>" . $jsondata;

   $myfile = fopen("westeros_ladies.txt", "w") or die("Unable to open file!");
   fwrite($myfile, $jsondata);
   fclose($myfile);

   echo '<br>' . 'Done writing file...';
?>
```

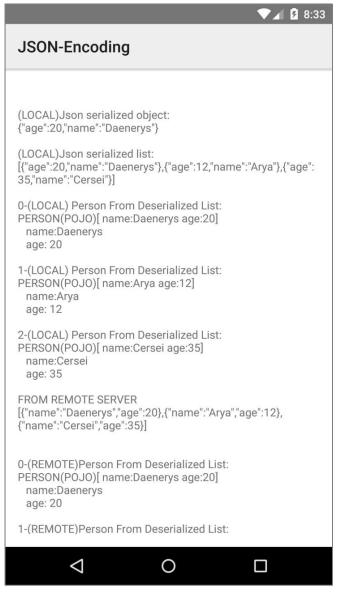
This server side PHP program writes to disk a JSON encoded data set.

Our Android app reads the data set, decodes it and creates an equivalent List<Person> object.

JSON Encoded Data
[{"name":"Daenerys","age":20},{"name":"Arya","age":12},{"name":"Cersei","age":35}]
Done writing file...

Reading JSON Data

Example 7. Using JSON & Android



Assume a server side app (similar to the PHP program depicted earlier) has created a GSON encoded data set. The set represents an array of Person objects. Each Person object includes **name** and **age.**

Our Android app connects to the server, downloads the file, and decodes it. The retrieved data is represented as a collection of Java Person objects held in a List<Person> collection.

Reading JSON Data

Example 7. Using JSON & Android

GSON is an implementation of JSON developed by Google. A user's guide for GSON is available from: https://sites.google.com/site/gson/gson-user-guide

To incorporate GSON to an Android app you need to follow the steps below:

1. Download the latest GSON API. Use the following MAVEN repository link: http://mvnrepository.com/artifact/com.google.code.gson/gson

2. Look under the "Gradle" tab for the name of the current release. At the time of writing its value is: **'com.google.code.gson:gson:2.4'**

   You will use it later to update the definition of the app's 'build.gradle'.

Reading JSON Data

Example 7. Using JSON & Android

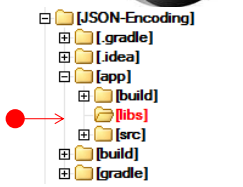
3. (Android Studio) Copy the downloaded *gson* jar to the app's **/libs** folder.

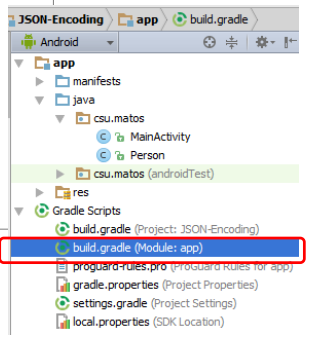
4. Add a reference to the gson jar in the module's build.gradle file as suggested in the following code:

```
...
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.google.code.gson:gson:2.4'
}
```

5. Click on the "Sync Now" link to update all the related Gradle files

Sync Now





Reading JSON Data

Example 7. Using JSON & Java - Layout

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
             xmlns:tools="http://schemas.android.com/tools"
             android:layout_width="match_parent" android:layout_height="match_parent"
             android:background="@android:color/white">

    <ProgressBar
        android:id="@+id/progressBar"
        android:layout_width="100dp"   android:layout_height="100dp"
        android:layout_gravity="center_horizontal"/>

    <LinearLayout
        android:layout_width="match_parent"    android:layout_height="match_parent"
        android:backgroundTint="@android:color/transparent"
        android:orientation="vertical">


        <ScrollView
            android:id="@+id/scrollView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">

            <TextView
                android:id="@+id/txtMsg"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"/>
        </ScrollView>

    </LinearLayout>
</FrameLayout>
```

Reading JSON Data

Example 7. Using JSON & Java - MainActivity 1 of 6

```
public class MainActivity extends Activity {
  ProgressBar progressBar;
  TextView txtMsg;
  Gson gson;

  Handler handler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
      super.handleMessage(msg);
      txtMsg.append("\n" + (String) msg.obj);
      progressBar.setVisibility(View.INVISIBLE);
    }
  };

  Thread slowWorkerThread = new Thread() {

    @Override
    public void run() {
      super.run();
      //a little delay here...
      try {Thread.sleep(2000);} catch (InterruptedException e) { }

      String text = "";
      //PART2: JSON Encoding
      Person person0 = new Person("Daenerys", 20);
```


Reading JSON Data

Example 7. Using JSON & Java - MainActivity 2 of 6

```
      // convert Person (Java object) to JSON format
      // display it as a JSON formatted string
      Gson gson = new Gson();
      String json = gson.toJson(person0);
      text = "\n(LOCAL)Json serialized object:\n" + json;
      handler.sendMessage(handler.obtainMessage(1, (String) text));
      // create a few more Person objects
      Person person1 = new Person("Arya", 12);
      Person person2 = new Person("Cersei", 35);

      // place all Person objects in an ArrayList
      ArrayList<Person> lstPerson = new ArrayList<Person>();
      lstPerson.add(person0);
      lstPerson.add(person1);
      lstPerson.add(person2);

      // convert Java ArrayList to JSON string
      String jsonList = gson.toJson(lstPerson);
      text = "\n(LOCAL)Json serialized list:\n" + jsonList;
      handler.sendMessage(handler.obtainMessage(1, (String) text));

      // use Java reflection to find the list's type
      Type arrayPersonType = new TypeToken<ArrayList<Person>>(){}.getType();
      // deserialize JSON string representing the list of objects
      ArrayList<Person> lst2 = gson.fromJson(jsonList, arrayPersonType);
```

Reading JSON Data

Example 7. Using JSON & Java - MainActivity 3 of 6

```
6   // explore the Java ArrayList
    for(int i=0; i<lst2.size(); i++){
      Person p = lst2.get(i);
      text = "\n" + i + "-(LOCAL) Person From Deserialized List:\n" + p.toString()
         + "\n   name:" + p.getName()
         + "\n   age: " + p.getAge();
      handler.sendMessage(handler.obtainMessage(1, (String) text));
    }

    try {
      // using java.net.URL;
7     URL url = new URL("http://informatos.org/westeros/westeros_ladies.txt");
      //URL url = new URL("http://192.168.1.70/westeros/westeros_ladies.txt");
      // ------------------------------------------------------------------------
      // next statement reads the ENTIRE file (delimiter \A matches All input)
      // String text = new Scanner( url.openStream() ).useDelimiter("\\A").next();
      // ------------------------------------------------------------------------
      // scanning remote file one line at the time
      text ="";
      Scanner scanner = new Scanner(url.openStream());
      while (scanner.hasNext()) {

        text += scanner.nextLine() + "\n";
      }
      handler.sendMessage(handler.obtainMessage(1, "\nFROM REMOTE SERVER\n" + text));
```


Reading JSON Data

Example 7. Using JSON & Java - MainActivity 4 of 6

```
      // use Java reflection to find the list's type
      Type arrayPersonType3 = new TypeToken<ArrayList<Person>>(){}.getType();

      // deserialize JSON string representing the list of objects
8     ArrayList<Person> lst3 = gson.fromJson(text, arrayPersonType3);

      // explore the Java ArrayList
      for(int i=0; i<lst3.size(); i++){
        Person p = lst3.get(i);
        text = "\n" + i + "-(REMOTE)Person From Deserialized List:\n" + p.toString()
            + "\n   name:" + p.getName()
            + "\n   age: " + p.getAge();
        handler.sendMessage(handler.obtainMessage(1, (String) text));
      }

    } catch (java.io.IOException e) {
      handler.sendMessage(handler.obtainMessage(1, "ERROR: " + e.getMessage()));
    }

  }//run
};
```


Reading JSON Data

Example 7. Using JSON & Java - MainActivity 5 of 6

```
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    txtMsg = (TextView) findViewById(R.id.txtMsg);
    progressBar = (ProgressBar)findViewById(R.id.progressBar);

    gson = new Gson();

    slowWorkerThread.start();

  }//onCreate


}
```

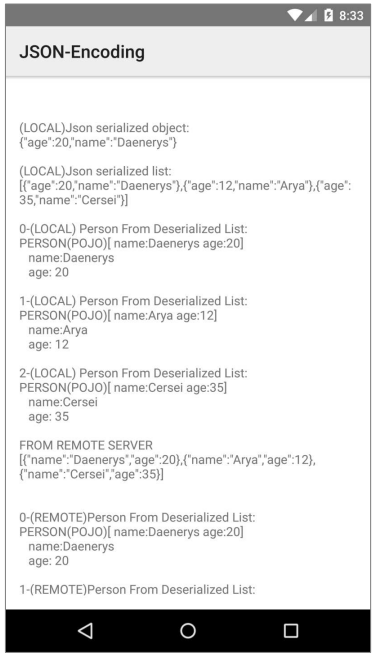

Reading JSON Data

Example 7. Using JSON & Java - MainActivity 6 of 6

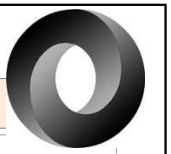
```
9   public class Person {
     private String name;
     private Integer age;

     public Person (String name, Integer age) {
        this.name = name; this.age = age;
     }
     public Person() {
        this.name = "n.a.";  this.age=0;
     };

     public String getName() {
        return name;
     }
     public void setName(String name) {
        this.name = name;
     }
     public Integer getAge() {
        return age;
     }
     public void setAge(Integer age) {
        this.age = age;
     }
     public String toString() {
        return "PERSON(POJO)=> name:" + name + " age:" + age;
     }
    }
```

Reading JSON Data

Example 7. Using JSON & Java



Reading JSON Data



Example 7. Using JSON & Java

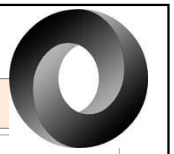
Comments

1. All the slow work is performed in a background Thread. First, a POJO (plain old java object) item of type **Person** is created.
2. The statement **`gson.toJson(person0)`** encodes the instance of person0 (comma separated items, inside curly-braces)
3. An **ArrayList<Person>** structure is created and populated with the instances of three person objects.
4. The **.toJson( )** method encodes the Java ArrayList<Person> object (comma separated objects inside braces)
5. You can use the GSON **TypeToken** class to find the generic type for a class. For example, to find the generic type for Collection<Foo>, you can use:

```
Type typeOfCollectionOfFoo = new TypeToken<Collection<Foo>>(){}.getType();
```

Assumes **Type** implements **equals()** and **hashCode().**

Reading JSON Data



Example 7. Using JSON & Java

Comments

6. The statement **.fromJson( )** uses the previously determined class type to properly decode the string representing the dynamic list of person objects.
7. The JSON data is regenerated as a common Java **ArrayList<Person>** class and traversed showing its contents.
8. **Person** is a POJO holding the attributes name and age, constructors, accessors, and a custom toString() method.
9. Observe the encoded JSON objects are exactly those previously seen in the PHP example (to be expedted - JSON is language independent)

Processing XML & JSON Data

< Questions />

References

http://developer.android.com/index.html
http://www.w3.org
http://www.saxproject.org/
https://code.google.com/p/google-gson/

Processing XML & JSON Data

Appendix A. Calculating Distance Between Two Coordinates

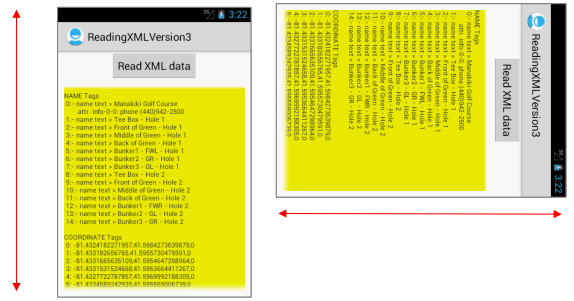
```
import android.location.Location;

. . .

private int distanceYards(GolfMarker gm){
    // calculating distance (yards) between
    // two coordinates
    int intDistance = 0;
    double distance = 0;

    Location locationA = new Location("point: Here");
    locationA.setLatitude(Double.parseDouble(aLatitude));
    locationA.setLongitude(Double.parseDouble(aLongitude));

    Location locationB = new Location("point: F/M/B Green");
    locationB.setLatitude(Double.parseDouble(bLatitude));
    locationB.setLongitude(Double.parseDouble(bLongitude));

    distance = locationA.distanceTo(locationB) * METER_TO_YARDS;
    intDistance = (int) Math.round(distance);
    return intDistance;
}
}// GolfMarker
```


Processing XML & JSON Data

Appendix B. Reminder – Keep your screen vertical !

NOTE:
For this Golf-GPS app you may want to modify the Manifest to stop (landscape) re-orientation. Add the following attributes to the <activity … > entry

```
android:screenOrientation="portrait"
android:configChanges="keyboardHidden|orientation"
```



Processing XML & JSON Data

Appendix C. Parsing a JSON Encoded String

The following fragments shows an alternative JSON decoding approach on which you traverse the underlining data structure looking for **jsonElements**, which could be: **jsonObject**, **jsonArray**, or **jsonPrimitive** tokens.

```
try {
  JsonElement jelement = new JsonParser().parse(jsonHouseStr);

  JsonObject jobject = jelement.getAsJsonObject();
  String departmenName= jobject.get("department").toString();
  String manager= jobject.get("manager").toString();
  System.out.println(departmenName + "\n" + manager);

  JsonArray jarray = jobject.getAsJsonArray("employeeList");

  for (int i = 0; i < jarray.size(); i++) {
    jobject = jarray.get(i).getAsJsonObject();
    String result = jobject.get("name").toString() + " "
                  + jobject.get("age").toString();

    System.out.println(" " + result);
  }

} catch (Exception e) {
  System.out.println(e.getMessage());
}
```

departmentName String	manager int	employeeList ArrayList<Person> (name, age)

Processing XML & JSON Data

Appendix C. Traversing Tree Structure of JSON Encoded Data

Example: The previous code fragment produces the following conversion

JSON encoded string	Equivalent Decoded Nodes
{"houseName":"Stark", "location":"Winterfell", "personLst":[{"name":"Catelyn Stark","age":40}, {"name":"Sansa Stark","age":14}, {"name":"Bran Stark","age":9}]}	"Stark" "Winterfell" "Catelyn Stark" 40 "Sansa Stark" 14 "Bran Stark" 9