Two important issues faced by applications securely exchanging data on the web include **data 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 and 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.
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:

```xml
<?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>
```
The main role of XML is to facilitate a transparent exchange of data over the Internet.


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

Consider the complex Employee class depicted here. Each node is an XML element. Id and Title are attributes of the class.
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.
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
<?xml version="1.0" encoding="utf8" ?>
<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>
  ...
</Employees>
```

Example taken from:
Microsoft XmlNotepad 2007
Example 1. Employee.xsd – Schema Definition (fragment)

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

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

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

<xsl:simpleType name="Zip">
  <xsl:restriction base="xs:positiveInteger">
    <xsl:maxInclusive value="99999" />
    <xsl:minInclusive value="00001" />
  </xsl:restriction>
</xsl:simpleType>
</xs:schema>
```
Reading XML Data

Example 2. Example: KML and Geographic Data

KeyHole Markup language (KML) is a file format used to display geographic data in an Earth Browser such as: Google Earth, Google Maps, and Google Maps for Mobile Apps.

Example 2. Example: KML and Geographic Data

In this example a Document consists of a set of Placemarks.

Each of our placemarks includes a name, description, and geolocation or point (a point consists of the values, latitude, longitude and [optionally] altitude).

Example 2. Mapping with KML (fragment)

```xml
<?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>
```
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?

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

Typical Distances for (Good) Amateur Players

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

Source: [http://golfcartoons.blogspot.com/] Golf Monthly 2009
Reading XML Data

Strategies for Reading/Parsing an XML File

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

<table>
<thead>
<tr>
<th>OPTION 1</th>
<th>OPTION 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A SAX (Simple API for XML)</td>
<td>W3C Document Builder</td>
</tr>
<tr>
<td>XmlPullParser</td>
<td></td>
</tr>
</tbody>
</table>

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.saxproject.org/)
- [http://www.w3.org/DOM/](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”.

14
In this example we will read an XML file saved in the app’s `/res/xml` folder. The file contains a set of KML placemarkers 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.

Inner attributes from an `<element>` can be extracted using the methods:
- `getAttributeCount()`
- `getAttributeName()`
- `getAttributeValue()`

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

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

<table>
<thead>
<tr>
<th>Attributes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AttributeName</td>
<td>AttributeValue</td>
</tr>
<tr>
<td>par</td>
<td>4</td>
</tr>
<tr>
<td>yards</td>
<td>390</td>
</tr>
</tbody>
</table>
Diagram showing the life-cycle of the `XmlPullParser` class. Any well-formed XML input document could be processed as suggested in the figure.
Example 4. SAX-Parsing a Resource XML File

Parsing the Tee-Time XML file listed below

```xml
<?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>
```
Example 4. SAX-Parsing of a Resource KML File

The same XMLPullParser used in the previous example is now used to dissect a KML file holding placemarker from a golf course.
<?xml version='1.0' encoding='UTF-8'?>
<kml xmlns='http://www.opengis.net/kml/2.2'>
  <Document>
    <gcPlace>
      <gcName>Manakiki Golf Course</gcName>
      <gcCity>Willoughby Hills</gcCity>
      <gcState>Ohio</gcState>
    </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>
Reading XML Data

Example 4. SAX-Parsing of a Resource KML File

```xml
<?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

Example 4. SAX-Parsing of a Resource KML File

```xml
<?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>
```

SAX
Simple API for XML

Example 4. SAX-Parsing of a Resource KML File

```xml
<?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

Example 4. SAX-Parsing of a Resource KML File

```java
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);
            }
        });
    }
}
```
Example 4. SAX-Parsing of a Resource KML File

```java
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
    }
}
```
@Override
protected StringBuilder doInBackground(Integer... params) {
    int xmlResFile = params[0];
    XmlPullParser parser = getResources().getXml(xmlResFile);
    StringBuilder stringBuilder = new StringBuilder();  
    String nodeText = "";
    String nodeName = "";
    try {
        int eventType = -1;
        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);
                stringBuilder.append(getAttributes(parser));
            } else if (eventType == XmlPullParser.END_TAG) {
                nodeName = parser.getName();
                stringBuilder.append("\nEND_TAG: " + nodeName);
            }
        }
    } catch (IOException e) {
        // Error handling
    } catch (XmlPullParserException e) {
        // Error handling
    }
    return stringBuilder.toString();
}
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
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++) {
            String attrName = parser.getAttributeName(i);
            String attrValue = parser.getAttributeValue(i);
            stringBuilder.append("\n    Attrib <key,value>= ", " + attrName + ", " + attrValue);
        }
    }
    return stringBuilder.toString();
} // getAttributes
}
// backgroundAsyncTask

} // ActivityMain
Reading XML Data

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.
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>, <coordinate>.
4. The parser will create Java-like lists to store all the text and attributes held in each node type.
5. For instance, our file is about a regulation golf course. 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’. The NodeList made for the <coordinate> elements contain the latitude and longitude of each entry in the <name> list.
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 XML element you request the construction of a NodeList (ArrayList-like collection) using the method: `.getElementsByTagName()`.

**PHASE 2.** Explore an individual node from a NodeList using the methods:
- `.item(i)`,
- `.getName()`,
- `.getValue()`,
- `.getFirstChild()`,
- `.getAttributes()`, etc.
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
<?xml version="1.0" encoding="utf-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Document>
    <course phone="(440)942-2500" length="6">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>
  </Document>
</kml>
```
After parsing the KML file shown in the previous image we obtain the following NodeLists.

The first arrows points to the entry <name> [2] which holds the value “Middle of the Green – Hole1”.

The second arrow point to its coordinates (<coordinate>[2] )
Reading XML Data

Example 5. The W3C DocumentBuilder Class

1. After scrolling the view shown in the previous image, we get the <course> information. Observe there is only one node with the tag <course> holding the text ‘Manakiki Golf Course’

2. The arrows point to <course> attributes: phone and length.
public class MainActivity extends Activity {
    private TextView txtMsg;
    Button btnGoParser;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_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);

                // this xml file includes elements: <course>,
                // <name>, <coordinates> for justa few holes
                String xmlFile = "manakiki_holes1and2.kml";
                // do slow XML reading in a separated thread
                new backgroundAsyncTask().execute(xmlFile);
            }
        });
    } // onCreate
} // onCreate
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) {
    String xmlFileName = params[0];
    return useW3CPARSER(xmlFileName);
}

// doInBackground
// backgroundAsyncTask

private String useW3CPARSER(String xmlFileName) {
    StringBuilder str = new StringBuilder();
    try {
        String kmlFile = Environment.getExternalStorageDirectory().getPath() + "/" + xmlFileName;
        InputStream is = new FileInputStream(kmlFile);
        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";
        }
        Document document = docBuilder.parse(is);
    } catch (Exception e) {
        Log.v("REALLY BAD!!!!", "document was NOT made by parser");
        return "BAD-ERROR";
    }
}
Example 5. The W3C DocumentBuilder Class

```java
// put all data into NodeLists
NodeList listNameTag = document.getElementsByTagName("name");
NodeList listCoordinatesTag = document.getElementsByTagName("coordinates");
NodeList listCourseTag = document.getElementsByTagName("course");

// traverse NodeLists for elements: <name>, <coordinates>, <course>.
str.append(getAllDataFromNodeList(listNameTag, "name"));
str.append(getAllDataFromNodeList(listCoordinatesTag, "coordinates"));
str.append(getAllDataFromNodeList(listCourseTag, "course"));

} 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
```
private Object getAllDataFromNodeList(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++) {
        Node node = list.item(i);
        int size = node.getAttributes().getLength();
        String text = node.getTextContent();
        str.append("\n   \" + i + ": " + text);

        // get all attributes of the current element (i-th hole)
        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. Identify the input XML file containing tag-elements: course, name, and coordinates. Do the potentially slow parsing of the file in a separated thread using an AsyncTask object, passing to it the SD-resident file’s name.

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>`.

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.