- XML Parsing
- JSON Parsing
- Connectivity Status
- HTTP Client
- Web Services

XML

Extensible Markup Language (XML) is a set of rules for encoding documents in a readable form.

- Similar to HTML but <tagElements> are user-defined.

- It is defined in the *XML Specification produced by the W3C.*

- XML's design goals emphasize transparency, simplicity, and transportability over the Internet.

- Example of XML-based languages include: RSS , Atom, SOAP, and XHTML.

XML

1.XML is used for **defining and documenting object classes.**

2.For example, an XML document (**.xml) might contain a collection of complex employee elements, such as <employee id="…" title="…" >...</employee> which lexically includes an "id" and "title" attributes.**

3.Employee may also hold other inner elements such as **"name", "country", "city", and "zip".**

4.An XML-Data schema (**.xsd) can describe such syntax.**

5. XML Tag define the scope of an element.

- Start Tag: Beginning of every non-element XML element
- End Tag: End tag should include solidus("/") before the name of the element
- XML tags are case-sensitive
- XML tags must be closed in an appropriate order

5. XML Comment

<!-------Your comment----->

XML-Parsing Standards

We will consider two parsing methods that implement W3C standards for accessing XML

- **SAX (Simple API for XML)**
  - event-driven parsing
  - "serial access" protocol
  - Read only API

- **DOM (Document Object Model)**
  - convert XML into a tree of objects
  - "random access" protocol
  - Can update XML document (insert/delete nodes)

XMLPullParser Interface

(Ref: www.xmlpull.org)

- XMLPullParser is an interface that defines parsing functionality.
- XmlPullParser(XPP) provides a simple and fast implementation of "pull parsing model" that allows processing application to request parsing events incrementally.
- There are two key methods: ***next() and nextToken().***
- While next() provides access to high level parsing events, nextToken() allows access to lower level tokens.

XMLPullParser

- The current event state of the parser can be determined by calling the **getEventType() method. Initially, the parser is in the START_DOCUMENT state.**
- The method next() advances the parser to the next event. The ***int value returned from next determines the current parser state and is identical to the value returned from following calls to getEventType().***
- The following event types are seen by next()
  - **START DOCUMENT** document start - parser has not yet read any input
  - **START_TAG** parser is on start tag
  - **TEXT** parser is on element content
  - **END_TAG** parser is on end tag
  - **END_DOCUMENT** document finished and no more parsing is allowed

XMLPullParser

- The current event state of the parser can be determined by calling the **getEventType() method. Initially, the parser is in the START_DOCUMENT state.**
- The method next() advances the parser to the next event. The ***int value returned from next determines the current parser state and is identical to the value returned from following calls to getEventType().***
- The following event types are seen by next()
  - **START DOCUMENT** document start - parser has not yet read any input
  - **START_TAG** parser is on start tag
  - **TEXT** parser is on element content
  - **END_TAG** parser is on end tag
  - **END_DOCUMENT** document finished and no more parsing is allowed

XMLPullParser

```
XmlPullParserFactory xmlFactoryObject = XmlPullParserFactory.newInstance();
myParser = xmlFactoryObject.newPullParser();
myParser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES, false);
myParser.setInput(in, null);
QuestionBank objQB = new QuestionBank();  Question objQs = null; String txtVars = "";
int event = myParser.getEventType();
while (event != XmlPullParser.END_DOCUMENT) {
    String tagname = myParser.getName();
    switch (event) {
    case XmlPullParser.START_TAG:
        if (tagname.equals("questionare")) {
          int duration ;    try{  duration=Integer.parseInt(myParser.getAttributeValue(null, "duration"));}
              catch (Exception e)   {  duration = 0 ;          }
          objQB.duration = duration ;  Log.d("Questionaire duration:",":"+duration );   }
          if (tagname.equals("Question")) {
                        objQs = new Question();
                        objQs.Qnumber = Integer.parseInt(myParser.getAttributeValue(null, "number").toString());   }  break;
    case XmlPullParser.TEXT:  txtVars = myParser.getText(); break;
    case XmlPullParser.END_TAG: {  if (objQs != null) {if (tagname.equals("topic"))  objQs.topic = txtVars;
                                                        if (tagname.equals("QStr"))  objQs.QStr = txtVars;
                                                        if (tagname.equals("Item1"))  objQs.Item1 = txtVars;
                                                        if (tagname.equals("Item2")) objQs.Item2 = txtVars;
                                                        if (tagname.equals("Item3")) objQs.Item3 = txtVars;
                                                        if (tagname.equals("Item4"))  objQs.Item4 = txtVars;
                                                        if (tagname.equals("Answer"))  objQs.Answer = Integer.parseInt(txtVars);
                                                        if (tagname.equals("Question"))  objQB.QestB.add(objQs);
                                        } } }           event = myParser.next();
        }
```

JSON

- JSON stands for **J**ava**S**cript **O**bject **N**otation
- JSON is lightweight text-data interchange language
- Data types:
  - Four basic built-in: strings, numbers, booleans (i.e true and false) and null.
  - Two structured data types - objects and arrays.
- An object in JSON is modeled using ***{..}***, while its attributes can be modeled using ***name : value*** pair.
  - Names are unique, non-null strings
  - Value can be an object, an array or a “simple” value, like a primitive value (int, String, boolean and so on)
- An array in JSON is represented using [..]. An array element can be an object, an array and simple value
- Both objects and arrays can be nested.

JSON Example

- {
  "employees": [
  { "firstName":"John" , "lastName":"Doe" },
  { "firstName":"Anna" , "lastName":"Smith" },
  { "firstName":"Peter" , "lastName":"Jones" }
  ] }

  This is JSON object of employee having array of three employees

- Android Classes for JSON
  - JSONObject
  - JSONArray

JSONObject

- The first step is to view the input string, categorize whether it is JSON Object or Array
- For parsing the JSON, create an object of class JSONObject/JSONArray from the input string
- This JSON object is collection of name/value mapping
- This class can look up both mandatory and optional values:
  - Use getType() to retrieve a mandatory value. This fails with a JSONException if the requested name has no value or if the value cannot be coerced to the requested type.
  - Use optType() to retrieve an optional value. This returns a system- or user-supplied default if the requested name has no value or if the value cannot be coerced to the requested type.
- Use put method to add value in JSONObject

JSONArray

- Values may be any mix of JSONObjects, other JSONArrays, Strings, Booleans, Integers, Longs, Doubles, null or NULL
- Use get/opt for getting object at particular index eg

  get(int index),

  getJSONObject(int index)
- Use put method to add value in JSONArray

JSON Parsing Example

```
{        try {
        String str1 = "{\"employees\": [\t{ \"firstName\":\"John\" , \"lastName\":\"Doe\" }, \t{
\"firstName\":\"Anna\" , \"lastName\":\"Smith\" }, \t{ \"firstName\":\"Peter\" , \"lastName\":\"Jones\" }\t]   }";
        Log.d("JSON String:", str1);
        JSONObject jso = new JSONObject(str1);
        JSONArray jsa = jso.getJSONArray("employees");
        for (int i=0; i< jsa.length(); i++)
        {
           JSONObject jso_details = jsa.getJSONObject(i);
           Log.d("JSON Details", jso_details.toString());
           String fName = jso_details.getString("firstName");
           String lName = jso_details.getString("lastName");
           Log.d("JSON Details", fName+":"+lName);
        }
      }    catch (Exception e)
      {
        Log.d("JSON String:", e.getMessage());
      }    }
```

Network Storage

Cloud Storage

Webserver Storage

Why we need network Storage ?

- We want to offloads the data gathering , transformation and storage to a powerful server.
- Can do a lot of things in very short time.
- It is connected to a large internet pipe.
- Your precious data will secure on network storage, if unfortunately your mobile lost.

❖ Here , we will focus on working on webserver storage with the help of webservices.

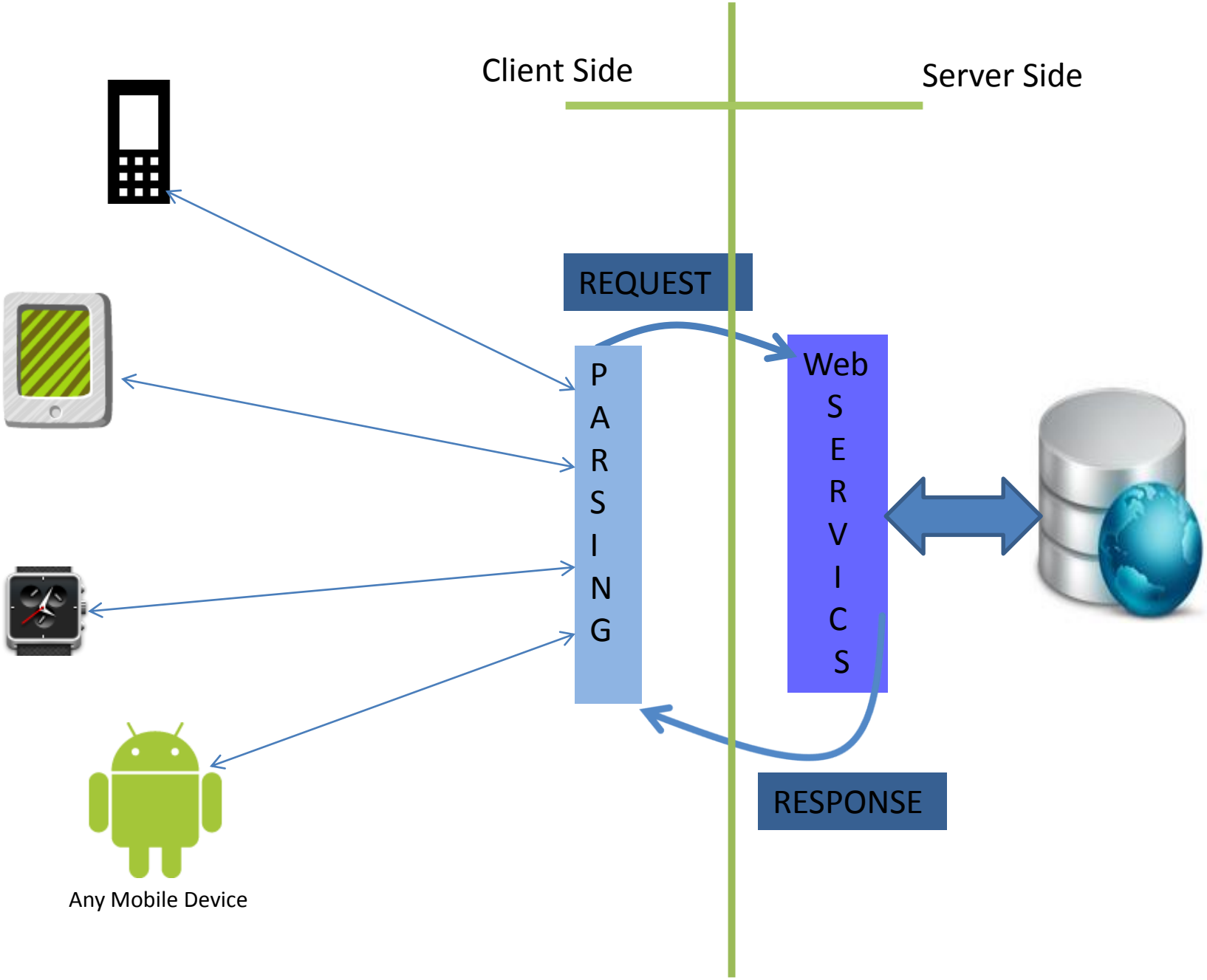
Web server Storage

- Web Server Storage is used with **web services**.
- Web services are method of communication beween client and server.
- Web services gives data which would not be in specific format for showing i.e. means we get xml /json formed which was just like on sever.
- Client side → your android mobile
  Server side → your own server
  data stored → **xml , json** or other formats

About webServices: web services are just like Q (Request) & A (Response). There are 2 types widely used.

1) REST web services:
- REQUEST: HTTP,XML,json or other;
- RESPONSE: XML, json or any format;

2) SOAP web services:
- REQUEST: XML, specific format;
- RESPONSE: XML, specific format;


Client Side
Server Side
REQUEST
PARSING
Web SERVICES
RESPONSE
Any Mobile Device

Data Network Connectivity Status

- **ConnectivityManager** is used to query active network and determine status.

```
ConnectivityManager cm =
        (ConnectivityManager)context.getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo activeNetwork = connMgr.getActiveNetworkInfo();
If activeNetwork is  not null -> Device is connected with Internet
One may need to find out, what type of network (Mobile or Wifi)
```

- Permission Required

```
<uses permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

- ConnectivityManager also broadcasts the change in connectivity details with action named **android.net.conn.CONNECTIVITY_CHANGE**

```
<action android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
```

Data Network Connectivity Status

```
ConnectivityManager connMgr = (ConnectivityManager)this.getSystemService(Context.CONNECTIVITY_SERVICE);
 NetworkInfo activeNetwork = connMgr.getActiveNetworkInfo();
if (activeNetwork != null) { // connected to the internet
            if (activeNetwork.getType() == ConnectivityManager.TYPE_WIFI) {
               // connected to wifi
               Toast.ma keText(this, activeNetwork.getTypeName(), Toast.LENGTH_SHORT).show();
            } else if (activeNetwork.getType() == ConnectivityManager.TYPE_MOBILE) {
               // connected to the mobile provider's data plan
               Toast.makeText(this, activeNetwork.getTypeName(), Toast.LENGTH_SHORT).show();
            }
         }
 else
 {
          AlertDialog.Builder alertBuilder = new AlertDialog.Builder(ListQuesPaper.this);
          alertBuilder.setTitle("Warning");
          alertBuilder.setMessage("No Internet Connection Available");
          alertBuilder.setPositiveButton("Ok", new DialogInterface.OnClickListener()  {
          @Override
          public void onClick(DialogInterface dialog, int which) {
          Toast.makeText(ListQuesPaper.this, "Positive Confirmation", Toast.LENGTH_LONG).show();
          dialog.cancel();  }; });

          AlertDialog ad = alertBuilder.create();
          ad.show() ;
 }
```

REST WebServices

- REST stands for **Re**presentational **S**tate **T**ransfer. (It is sometimes spelled "ReST".) It relies on a stateless, client-server, cacheable communications protocol -- and in virtually all cases, the HTTP protocol is used
- RESTful applications use HTTP requests to post data (create and/or update), read data (e.g., make queries), and delete data. Thus, REST uses HTTP for all four CRUD (Create/Read/Update/Delete) operations.
- REST is a lightweight alternative to mechanisms like RPC (Remote Procedure Calls) and Web Services (SOAP, WSDL, et al.)
- REST is not a "standard". There will never be a W3C recommendataion for REST

http://rest.elkstein.org/

Establishing HTTP Connection

- Android supports two HTTP clients:
  - Java.net.HttpURLConnection
  - Apache HttpClient (org.apache.http.impl.client.DefaultHttpClient)
- Apache HTTP client has fewer bugs on Eclair and Froyo. It is the best choice for these releases.
- For Gingerbread and better, HttpURLConnection is the best choice. Its simple API and small size makes it great fit for Android. Transparent compression and response caching reduce network use, improve speed and save battery.
- New applications should use HttpURLConnection;

Calling REST WebServices - DefaultHttpClient

```
DefaultHttpClient httpClient = new DefaultHttpClient();
HttpPost httpPost = new HttpPost(url); // or HttpGet
httpPost.setEntity(new UrlEncodedFormEntity(params));
HttpResponse httpResponse = httpClient.execute(httpPost);
HttpEntity httpEntity = httpResponse.getEntity();
String st = EntityUtils.toString(httpEntity, HTTP.UTF_8);
```

Getting data with REST WebServices

It is popular and widely used web services in android.
REST services are more ad-hoc than SOAP services since they don't use WSDL and they rely on pre-established standards (ex. XML,json and HTTP,etc).

```
HttpClient httpclient = new DefaultHttpClient();
HttpResponse response;
String responseString = null;
try {
   response = httpclient.execute(new HttpGet(uri[0]));
   StatusLine statusLine = response.getStatusLine();
   if(statusLine.getStatusCode() == HttpStatus.SC_OK){
      ByteArrayOutputStream out = new ByteArrayOutputStream();
      response.getEntity().writeTo(out);
      out.close();
      responseString = out.toString();
   } else{
      //Closes the connection.
      response.getEntity().getContent().close();
      throw new IOException(statusLine.getReasonPhrase());
   }
} catch (ClientProtocolException e) {
   e.printStackTrace();
} catch (IOException e) {
   e.printStackTrace();
}
```

Output:
Here we get output in the formd on server.

Calling REST WebServices - **HttpURLConnection**

- Uses of this class follow a pattern:
- Create URL by appending parameters (eg http://android.sisoft.in/sellerApp/...)
- Obtain a new **HttpURLConnection** by calling **URL.openConnection()** and casting the result to **HttpURLConnection**.
- Prepare the request. Request headers may also include metadata such as credentials, preferred content types, and session cookies.
  - Optionally upload a request body. Instances must be configured with **setDoOutput(true)** if they include a request body. Transmit data by writing to the stream returned by **getOutputStream()**
- Read the response. Response headers typically include metadata such as the response body's content type and length, modified dates and session cookies. The response body may be read from the stream returned by **getInputStream()**. If the response has no body, that method returns an empty stream.
- Disconnect. Once the response body has been read, the HttpURLConnection should be closed by calling **disconnect()**. Disconnecting releases the resources held by a connection so they may be closed or reused.

HttpURLConnection – URL Object

- URL object may be created by passing string URI to URL Constructor
  - URL url = **new** URL(**str_url**);
- This may be string uri may be built by concating different arguments
- Or This may done by URI.Builder class
  - Uri.Builder urib = **new** Uri.Builder();

```
urib.scheme("http")
        .authority("android.sisoft.in")
        .appendPath("seller")
        .appendPath("orders_assigntome.php")
        .appendQueryParameter("OrderID", ordNum)
        .appendQueryParameter("UserID", uid)
        .appendQueryParameter("locLAT", locLat)
        .appendQueryParameter("locLONG", locLang);

str_url = urib.build().toString();
```

HttpURLConnection – Request Buffer Handling

- Set Request method for Connection using methods
  - urlConn.setRequestMethod(“GET”) // default is GET
  - urlConn.setRequestMethod(“POST”) ;
- In case of Post method, parameter must be sent along with Request Buffer. For this, URLConnection must open output stream.
  - urlConn.setOutput(true)
- Format the post parameters in string in key=value pair connected by “&” sign
  - String postParameters = “id=302&num=1234567"
- Open output stream and preferable change this to BufferedOutputStream and write the post parameter on this outputStream
- The output stream must be flushed and closed.

HttpURLConnection – Response Buffer Handling

- Read the Response Code
  - int responseCode = urlConn.getResponseCode();
- Accept the input data coming from server. For this connection must signal this using setDoInput(true)
  - urlConn.setDoInput(true);
- Open the inputsteam and read the data coming from server

- *// Read Server Response*

```
InputStream inputStream = httpURLConnection.getInputStream();
bufferedReader = new BufferedReader(new InputStreamReader(inputStream,
"iso-8859-1"));
String response = "";
String line = "";
while ((line = bufferedReader.readLine()) != null) {
   response += line;
}
bufferedReader.close();
inputStream.close();
```

Establishing HTTP Connection

```
URL url = new URL(params[0]+"/"+params[1]);
HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
urlConnection.setRequestMethod("GET");
urlConnection.setDoOutput(true);
// connect
urlConnection.connect();
String fileLocation = new String(Environment.getExternalStorageDirectory() + "");
Log.d("SDCard File Path::", fileLocation);
file = new File(fileLocation, params[1]); // , "android007.xml");
FileOutputStream fileOutput = new FileOutputStream(file);

InputStream inputStream = urlConnection.getInputStream();
// this is the total size of the file which we are downloading
totalSize = urlConnection.getContentLength();

// create a buffer...
byte[] buffer = new byte[1024];
int bufferLength = 0;

while ((bufferLength = inputStream.read(buffer)) > 0) {
fileOutput.write(buffer, 0, bufferLength);
downloadedSize += bufferLength;


int per = (int) ((downloadedSize *100) / totalSize);
publishProgress(per);
Log.d("Download in Backrground", per+" ");

}
```

SOAP

- **S**imple **O**bject **A**ccess **P**rotocol
- Format for sending messages over Internet between programs
- XML-based
- Platform and language independent
- Simple and extensible
- Stateless, one-way
  - But applications can create more complex interaction patterns

SOAP Building Blocks

- Envelope (required) – identifies XML document as SOAP message
- Header (optional) – contains header information
- Body (required) –call and response information
- Fault (optional) – errors that occurred while processing message

WSDL (Web Service Description Language)

- Standard method of describing Web Services and their capabilities
- Idea: Automate details involved in applications communication
- Operations and messages are described abstractly
- Bound to a concrete network protocol and message format to define an endpoint
- Provide documentation for distributed systems

WSDL Details

- A WSDL document defines **services**
- Services are collection of network endpoints (**ports)**
- Abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings
- Allows the reuse of abstract definitions:
  - **messages -**abstract descriptions of data being exchanged
  - **port types** -abstract collections of **operations**
  - concrete protocol and data format specifications for a particular port type constitutes a reusable **binding**

Soap Web Services in Android: kSoap2

- On Android, consuming a SOAP web service is more difficult (as opposed to consuming a RESTful web service), since it does not include any libraries to communicate with SOAP web services
  - For bridging this gap, download a library jar file named **kSOAP2** from following link: http://code.google.com/p/ksoap2-android/source/browse/m2-repo/com/google/code/ksoap2-android/ksoap2-android-assembly/2.5.4/ksoap2-android-assembly-2.5.4-jar-with-dependencies.jar
  - Right click on "View Raw File" and "Save Link As…"). Save file along with project source.
  - Copy the jar file in project/libs
  - Project-> Properties-> Library-> Add Jars

Various classes of kSOAP2 : (Ref: http://ksoap2.sourceforge.net/doc/api/)

- SoapObject: A simple dynamic object that can be used to build soap calls. This goes inside the body of a soap envelope
- PropertyInfo: This class is used to store information about each property an implementation
- SoapSerializationEnvelope: This class extends the SoapEnvelope with Soap Serialization functionality.
- SoapPrimitive: A class that is used to encapsulate primitive types
- HttpTransportSE: A J2SE based HttpTransport layer.

Steps for Invoking Soap Web Services

- Create a Services Request
  - All SOAP requests are created in kSOAP2 using a org.ksoap2.serialization.SoapObject object. You need to specify the web service namespace, and the service method name when constructing a SoapObject. You can find the namespace and method name details in the wsdl file itself, or in the documentation provided by the service provider.
- Add request properties to SoapObject using PropertyInfo class
  - org.ksoap2.serialization.PropertyInfo class enables to add property name-value pairs
- **Create a SOAP Envelope** using the org.ksoap2.serialization.SoapSerializationEnvelope class.
- Add the service request to Soap Envelope using setOutputSoapObject(Soap Object)
- **Create an HTTP Transport request object** to deliver the SOAP request (org.ksoap2.transport.HttpTransportSE)
- **Send the SOAP request over HTTP** using the HTTP Transport and SOAP Envelope objects created earlier

```
public class StockQuoteFetcher {
   private final String NAMESPACE = "http://www.webserviceX.NET/";
   private final String METHOD_NAME = "GetQuote";
   private final String SOAP_ACTION = "http://www.webserviceX.NET/GetQuote";
   private final String URL = http://www.webservicex.net/stockquote.asmx
   private final SoapSerializationEnvelope envelope;
         public StockQuoteFetcher(String quotes)
         {   SoapObject request = new SoapObject(NAMESPACE, METHOD_NAME);
            PropertyInfo quotesProperty = new PropertyInfo();
            quotesProperty.setName("symbol");
            quotesProperty.setValue(quotes);
            quotesProperty.setType(String.class);
            request.addProperty(quotesProperty);
            envelope = new SoapSerializationEnvelope(SoapEnvelope.VER11);
            envelope.dotNet = true;
            envelope.setOutputSoapObject(request);
          }
                  public String Fetch()
                  {
                     String result = "";
                     HttpTransportSE  httpRequest = new  HttpTransportSE(URL);
                     try
                     {
                        httpRequest.call(SOAP_ACTION, envelope);
                        SoapPrimitive response = (SoapPrimitive)envelope.getResponse();
                        result =  response.toString();
                     }
                     catch(Exception e)    { e.printStackTrace();   }
                     return result;
                  }
```

```
public class StockList extends ListActivity {

   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      List<StockQuote> stocks = new ArrayList<StockQuote>();
      stocks.add(new StockQuote("MSFT", 24.78));
      stocks.add(new StockQuote("ORCL", 34.02));
      stocks.add(new StockQuote("AMZN", 180.13));
      stocks.add(new StockQuote("ERTS", 19.73));

      StockQuoteFetcher sqf = new StockQuoteFetcher(stocks.get(0)
                                    .getTickerSymbol());
      String xmlResult = sqf.Fetch();
      stocks.add(new StockQuote(xmlResult, 0.0));

      setListAdapter(new StockQuoteAdapter(this, stocks));
   }
}
```

Now , next you have to parse to the fetched data. Show it as an easier form.
You will see that on further pages.