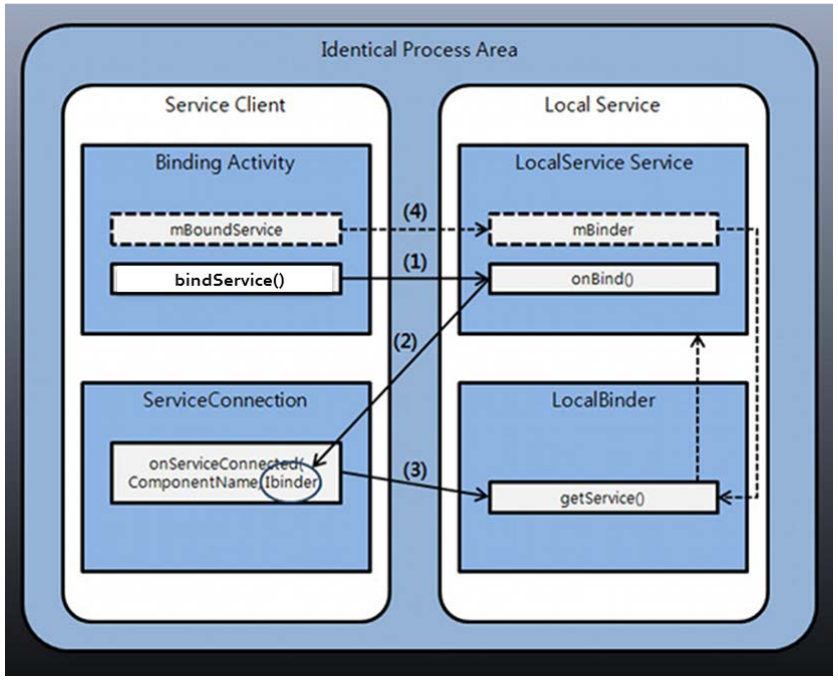
Network Programming

Programming the Android Platform

CS 282
Principles of Operating Systems II
Systems Programming for Android

Overview of Network Programming

- Android provides various mechanisms for local IPC between processes
  - e.g., Messengers, AIDL, Binder, etc.



- Many Android apps provide & use data & services via the Internet
  - e.g., Browser, Email, Calendar, etc.

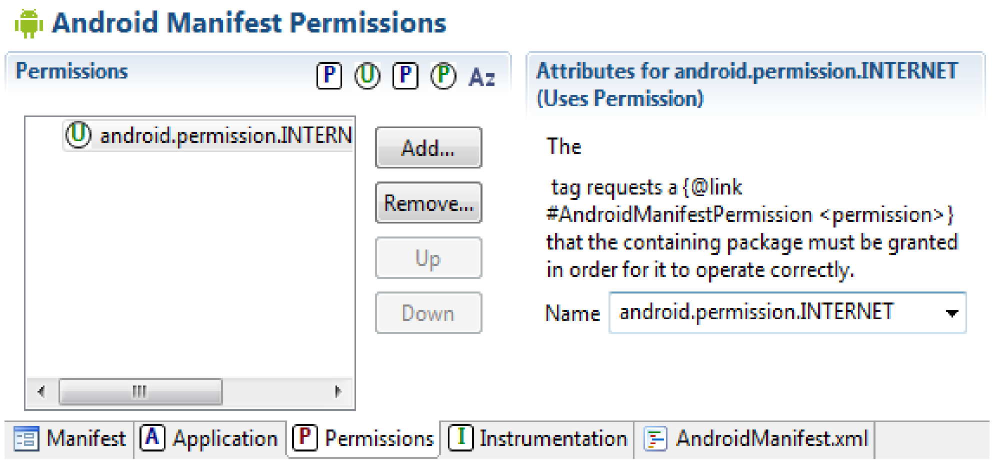
- Programming robust, extensible, & efficient networked software is hard
  - e.g., must address many complex topics that are less problematic for stand-alone apps

Android Network Programming

- Android includes multiple network programming classes, e.g.,
  - java.net – (Socket, URL)
  - org.apache - (HttpRequest, HttpResponse)
  - android.net – (URI, AndroidHttpClient, AudioStream)

Networking Permissions

- To allow an app to access the Internet using Eclipse, open AndroidManifest.xml, go to “Permissions” tab, add "Uses Permission" & select android.permission.INTERNET



Attributes for android.permission.INTERNET (Uses Permission)

The

tag requests a {@link #AndroidManifestPermission <permission>} that the containing package must be granted in order for it to operate correctly.

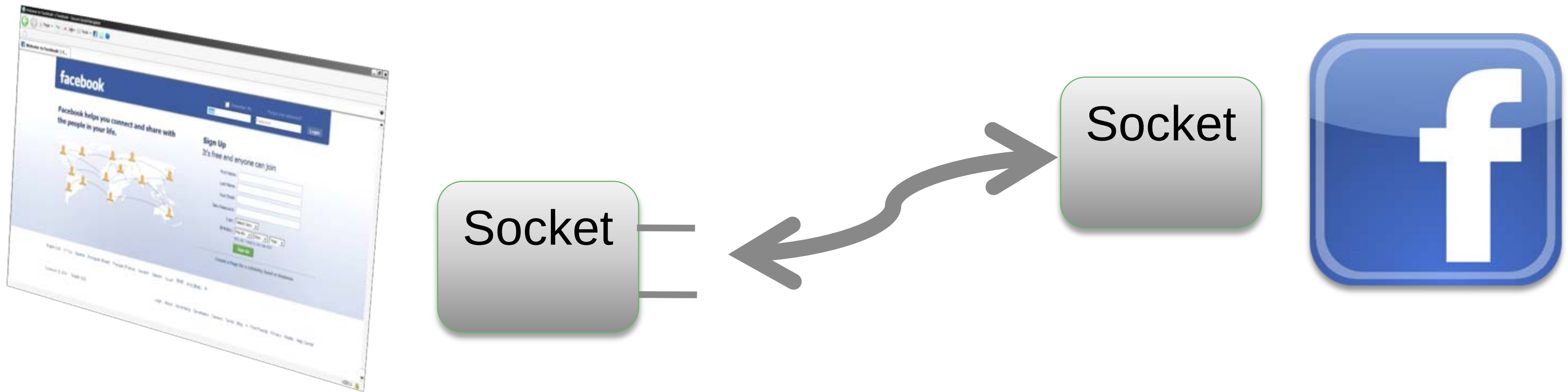
Name: android.permission.INTERNET

- Alternatively, open the file as raw XML & add the following line right after the <uses-sdk… /> element: <uses-permission android:name="android.permission.INTERNET"/>
- If you don't do this, your application will crash with an UnknownHostException when trying to access a remote host

http://ffct.cc/android-development-accessing-the-internet

Overview of Sockets

- A socket is a software endpoint that can create a bi-directional "reliable" communication link between software processes
- Sockets are a common programming interface for performing network communication
- Underneath the hood, Android's HTTP client library uses Java sockets to send & receive data



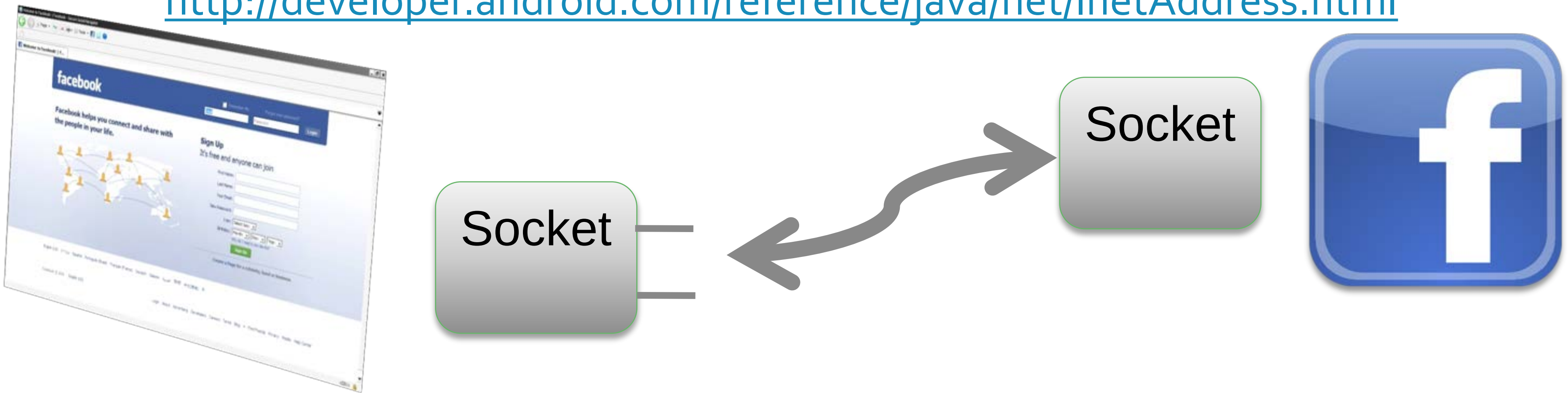
Overview of Java Sockets

- There are three main classes in Java Sockets
  - ServerSocket – This class represents a server-side socket that waits for incoming client connections

    http://developer.android.com/reference/java/net/ServerSocket.html
  - Socket – Provides a client-side TCP socket

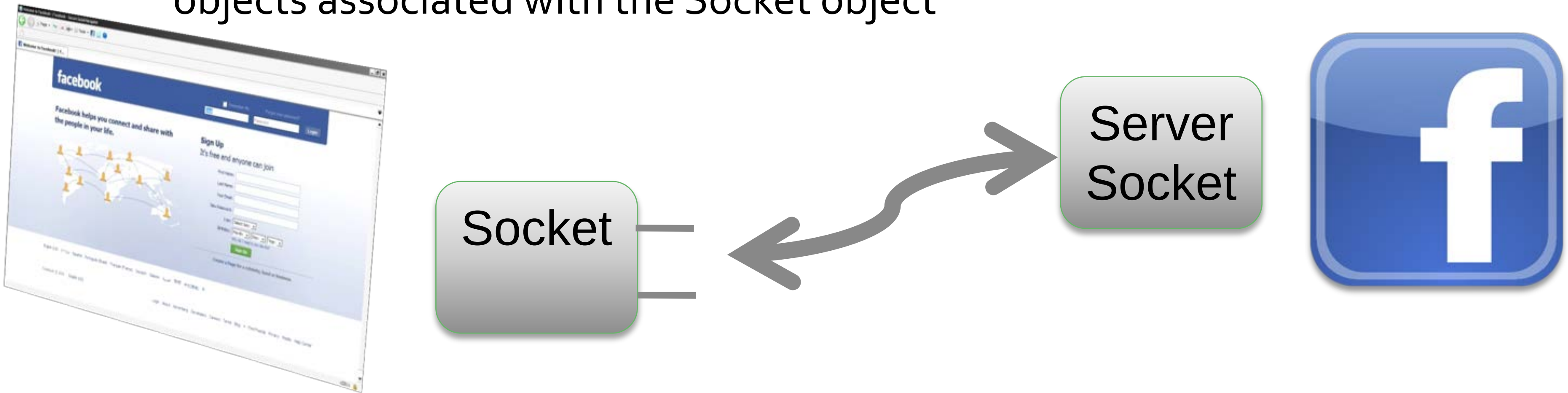
    http://developer.android.com/reference/java/net/Socket.html
  - InetAddress – Defines an Internet Protocol (IP) address

    http://developer.android.com/reference/java/net/InetAddress.html



Overview of Java Sockets (cont'd)

- In Java, a ServerSocket can receive new connections from a client
  - Plays the role of the "Acceptor" in the Acceptor/Connector pattern
- A client connects to a remote ServerSocket through a data-mode Socket instance
- When the server receives the connection, it talks to the client using a data-mode socket
  - The actual exchange of bytes is done via InputStream & OutputStream objects associated with the Socket object



InputStream in Java

- An InputStream is a stream of incoming byte data
- An InputStream can be obtained from a Socket by using the getInputStream() method
- To read from a stream, you must create a byte buffer to read in data
- Each call to read on an InputStream fills your buffer with data & returns the number of bytes read

```
InputStream in = someSocket.getInputStream();
const int BUFSIZ = 1024;
byte[] buffer = new byte[BUFSIZ];

for(int bytesRead; (bytesRead = in.read(buffer,0,buffer.length)) != -1; ) {
   // the buffer has been filled, do something with the data
}
```

http://developer.android.com/reference/java/io/InputStream.html

InputStreamReader in Java

- An InputStreamReader turns a byte stream into a character stream
- Data read from the source input stream is converted into characters by either a default or a provided character converter
- InputStreamReader contains an 8K buffer of bytes read from the source stream & converts these into characters as needed

```
InputStream in = someSocket.getInputStream();
Reader reader = new InputStreamReader(in);

for (int data; (data = reader.read()) != -1; ){
  char theChar = (char) data;
   // … do something with the data
}
reader.close();
```

http://developer.android.com/reference/java/io/InputStreamReader.html

BufferedReader in Java

- Wraps an existing Reader & *buffers* the input
- Expensive interaction with underlying reader is minimized, since most (smaller) requests can be satisfied by accessing buffer alone
- Drawback is that some extra space is required to hold the buffer & copying takes place when filling that buffer

```
BufferedReader bufferedReader = new BufferedReader(new
     InputStreamReader(someSocket.getInputStream()));

for (String data; (data = bufferedReader.readLine()) != null; ){
   // … do something with the data
}

bufferedReader.close();
```

http://developer.android.com/reference/java/io/BufferedReader.html

OutputStream in Java

- An OutputStream is a stream of outgoing byte data
- An OutputStream can be obtained from a Socket by using the getOutputStream() method
- You can write data to a stream by passing in a byte buffer of data
- You should use the flush() method if you want to make sure that the data you have written has been output to disk or sent to the other end of the socket

```
OutputStream out = someSocket.getOutputStream();
out.write("Hello Socket".getBytes());
out.flush();
byte[] buffer = new byte[1024];
out.write(buffer,0,buffer.length); //fill the buffer
out.close();
```

http://developer.android.com/reference/java/io/OutputStream.html

OutputStreamWriter in Java

- A class for turning a character stream into a byte stream
- Data written to the target input stream is converted into bytes by either a default or a provided character converter
- OutputStreamWriter contains an 8K buffer of bytes to be written to target stream & converts these into characters as needed

```
OutputStreamWriter out = new OutputStreamWriter
    (someSocket.getOutputStream());
String string1 = ”Socket IO”, string2 = ” is fun”;
out.write(string);
out.append(string);
out.flush();
out.close();
```

http://developer.android.com/reference/java/io/OutputStreamWriter.html

PrintWriter in Java

- Wraps either an existing OutputStream or an existing Writer (including OutputStreamWriter)
- Provides convenience methods for printing common data types in a human readable format

```
 StringBuffer data = new StringBuffer();
PrintWriter pw = new PrintWriter(new // “true” indicates “auto-flush”
    OutputStreamWriter(someSocket.getOutputStream()),true);
 pw.println("GET /index.html");

 BufferedReader br = new BufferedReader(new InputStreamReader(
                                                   socket.getInputStream()));
 String rawData;
 while ((rawData = br.readLine()) != null) { data.append(rawData); }
```

http://developer.android.com/reference/java/io/PrintWriter.html

Using Sockets

```
public class NetworkingSocketsActivity extends Activity {
 TextView mTextView = null;
  public void onCreate(Bundle savedInstanceState) {
    ...
    // assuming server at www.dre.vanderbilt.edu listening on port 80
    new HttpGet().execute("www.dre.vanderbilt.edu ");
  }

  // Display the text on the screen
  private void onFinishGetRequest(String result) {
    mTextView.setText(result);
}
```

Using Sockets (cont.)

```
private class HttpGet extends AsyncTask<String, Void, String>
{
protected String doInBackground(String... params) {
    Socket socket = null;
    StringBuffer data = new StringBuffer();
    try {
      socket = new Socket(params[0], 80);
      PrintWriter pw = new PrintWriter(
          new OutputStreamWriter(socket.getOutputStream()),
                                                    true);
      pw.println("GET /index.html");
     ...
```

Using Sockets (cont.)

```
      ...
      BufferedReader br = new BufferedReader(
          new InputStreamReader( socket.getInputStream()));
      String rawData;
      while ((rawData = br.readLine()) != null) {
        data.append(rawData);
      }
    } catch ...
        return data.toString(); // close socket  ...
  }
  protected void onPostExecute(String result) {
    onFinishGetRequest(result);
  }
...
```

HTTP Clients in Android

- Android includes two HTTP clients: HttpURLConnection & Apache HTTP Client
  - Both support HTTPS, streaming uploads & downloads, configurable timeouts, IPv6 & connection pooling
- Apache HTTP client has fewer bugs in Android 2.2 (Froyo) & earlier releases
- For Android 2.3 (Gingerbread) & later, HttpURLConnection is the best choice
  - Its simple API & small size makes it great fit for Android
  - Transparent compression & response caching reduce network use, improve speed & save battery
- See the http://android-developers.blogspot.com/2011/09/androids-http-clients.html for a comparison of the two HTTP clients

HttpURLConnection

- Used to send & receive HTTP data of any type/length over the web
- This class may be used to send & receive streaming data whose length is not known in advance
- Uses of this class follow a pattern:
  1. Obtain new HttpURLConnection by calling URL.openConnection() & cast result
  2. Prepare the request
     - The primary property of a request is its URI
  3. Transmit data by writing to the stream returned by getOutputStream()
  4. Read response from stream returned by getInputStream()
  5. Once response body has been read, HttpURLConnection should be closed by calling disconnect(), which releases resources held by a connection so they may be closed or reused

http://developer.android.com/reference/java/net/HttpURLConnection.html

HTTP Client Classes

- Useful HTTPClient classes
  - AndroidHttpClient – Implementation of Apache DefaultHttpClient configured with reasonable default settings & registered schemes for Android

    http://developer.android.com/reference/android/net/http/AndroidHttpClient.html
  - HttpGet – Retrieves whatever information (in the form of an entity) is identified by the Request-URI
    http://developer.android.com/reference/org/apache/http/client/methods/HttpGet.html
  - ResponseHandler – Handler that encapsulates the process of generating a response object from a HttpResponse
    http://developer.android.com/reference/org/apache/http/client/ResponseHandler.html

Using URLConnection

```
public class NetworkingURLActivity extends Activity {
 TextView mTextView = null;
 public void onCreate(Bundle savedInstanceState) {
   new HttpGetTask().execute( "http://api.geonames.org/…");
 }
 …
```

Using URLConnection (cont.)

```
…
 private class HttpGetTask extends AsyncTask<String, Void, String> {
   protected String doInBackground(String... params) {
     StringBuffer data = new StringBuffer();
     BufferedReader br = null;
     try {
       HttpURLConnection  conn =  (HttpURLConnection) new
                                    URL(params[0]).openConnection();
      // read & process response
     }
     // close outputStream
     return data.toString();
}
…
```

Using HTTPClient

```
…
 private class HttpGetTask extends AsyncTask<String, Void,
   String> {
  protected String doInBackground(String... params) {
   AndroidHttpClient client = AndroidHttpClient.newInstance("");
   HttpGet request = new HttpGet(params[0]);
   ResponseHandler<String> responseHandler =
                                        new BasicResponseHandler();
   try {
     return client.execute(request, responseHandler);
   } catch (/* … */)
   return null;
 }
…
```

Parsing HTTP Responses

- The protocol underlying Java Sockets & HTTP is TCP/IP, which is a byte stream protocol
- It's therefore necessary to parse the HTTP responses to extract out the relevant data fields
- Several popular formats including
  - Javascript Object Notation (JSON)
  - eXtensible Markup Language (XML)

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<foo>Hello World!</foo>
```

JSON

JavaScript Object Notation

Javascript Object Notation (JSON)

- Javascript Object Notation
  - http://www.json.org/
- Intended as a lightweight data interchange format used to serialize & transmit structured data over a network connection
  - It is used primarily to transmit data between a server & web application
- Data packaged in two types of structures:
  - Maps of key/value pairs
  - Ordered lists of values

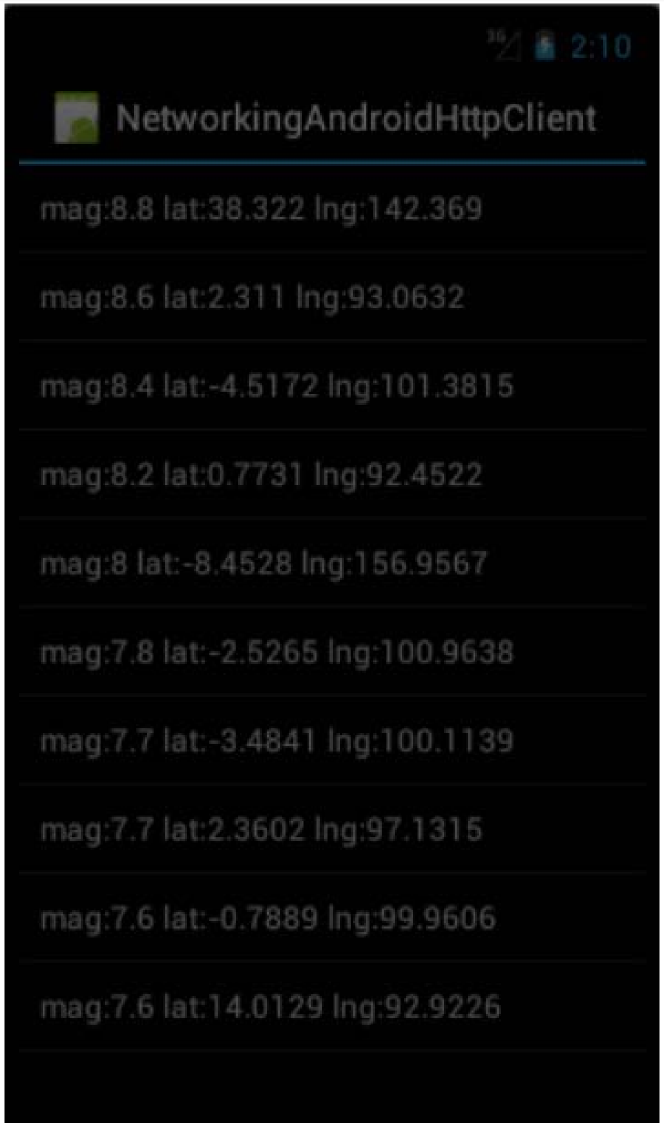
JSON
JavaScript Object Notation

- JSON's basic types are:
  - Number (double precision floating-point format)
  - String
  - Boolean (true or false)
  - Array (an ordered sequence of values, comma-separated & enclosed in square brackets)
  - Object (an unordered collection of key:value pairs with the ':' character separating (distinct string) key & value, comma-separated & enclosed in curly braces)
  - null (empty)

Earthquake Data (JSON Output)

- http://api.geonames.org/earthquakesJSON?north=44.1&south=-9.9&east=-22.4&west=55.2&username=demo
- Produces

```
{"earthquakes":
    [
    {"eqid":"c0001xgp","magnitude":8.8,"lng":142.369,"src":"us",
       "datetime":"2011-03-11 04:46:23","depth":24.4,"lat":38.322},
    {"eqid":"2007hear","magnitude":8.4,"lng":101.3815,"src":"us",
       "datetime":"2007-09-12 09:10:26","depth":30,"lat":-4.5172},
    ...
    {"eqid":"2010xkbv","magnitude":7.5,"lng":91.9379,"src":"us",
    "datetime":"2010-06-12 17:26:50","depth":35,"lat":7.7477}
     ]
}
```



```
mag:8.8 lat:38.322 lng:142.369
mag:8.6 lat:2.311 lng:93.0632
mag:8.4 lat:-4.5172 lng:101.3815
mag:8.2 lat:0.7731 lng:92.4522
mag:8 lat:-8.4528 lng:156.9567
mag:7.8 lat:-2.5265 lng:100.9638
mag:7.7 lat:-3.4841 lng:100.1139
mag:7.7 lat:2.3602 lng:97.1315
mag:7.6 lat:-0.7889 lng:99.9606
mag:7.6 lat:14.0129 lng:92.9226
```

Parsing JSON

```
class HttpGetTask extends AsyncTask<String, Void, List<String>>
{
  protected List<String> doInBackground(String... params) {
    AndroidHttpClient client = AndroidHttpClient.newInstance("");
    HttpGet request = new HttpGet(params[0]);
    JSONResponseHandler responseHandler =
                                        new JSONResponseHandler();
    try {
      return client.execute(request, responseHandler);
    } catch …
    return null;
  }
…
```

Parsing JSON (cont.)

```
class JSONResponseHandler implements ResponseHandler<List<String>> {
  public List<String> handleResponse(HttpResponse response) throws
                                              ClientProtocolException, IOException {
  List<String> result = new ArrayList<String>();
  String JSONResponse =
                    new BasicResponseHandler().handleResponse(response);
  try {
    JSONObject object =
          (JSONObject) new JSONTokener(JSONResponse).nextValue();
    JSONArray earthquakes = object.getJSONArray("earthquakes");
    for (int i = 0; i < earthquakes.length(); i++) {
      JSONObject tmp = (JSONObject) earthquakes.get(i);
      result.add("mag:" + tmp.get("magnitude") +
                    " lat:" + tmp.getString("lat") + " lng:" + tmp.get("lng"));
     }
  } catch (JSONException e) { … }
return result;
```

eXtensible Markup Language (XML)

- eXtensible Markup Language (XML) can structure, store, & transport data by defining a set of rules for encoding documents in format that is both human-readable & machine-readable
  - See http://www.w3.org/TR/xml
- XML documents are made up of storage units called entities
- Entities can contain markup
  - Markup encodes description of storage layout & logical structure of documents

- XML is a popular format for sharing data on the internet
- Websites that update their content often provide XML feeds so that external programs can keep abreast of content changes
- Uploading & parsing XML data is a common task for network-connected apps

```
<?xml version="1.0" encoding="UTF-8"?>
<foo>Hello World!</foo>
```

Google recommends XmlPullParser to parse XML on Android

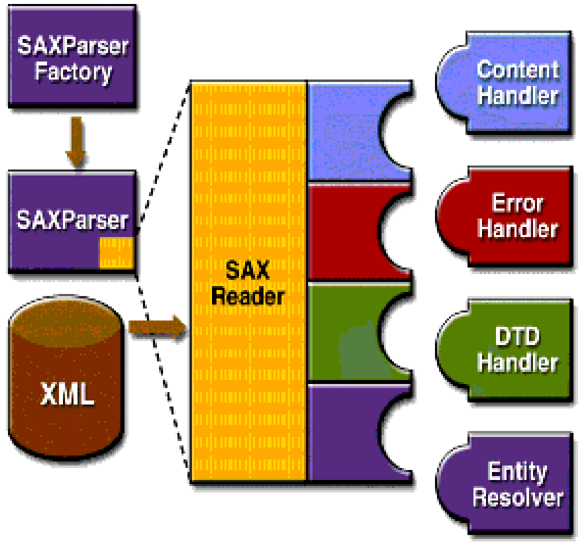
Parsing XML

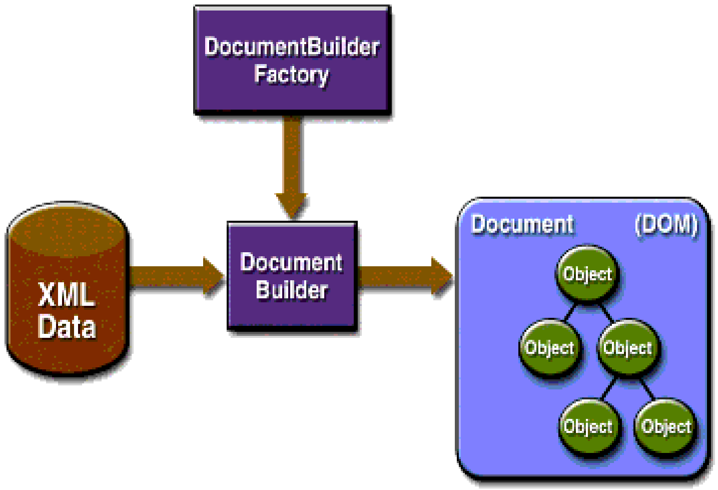
- Several types of XML parsers available
  - SAX – Streaming with application callbacks

    http://developer.android.com/reference/android/sax/package-summary.html
  - DOM – Coverts document into a tree of nodes & then can apply Visitor pattern

    http://developer.android.com/reference/org/w3c/dom/package-summary.html
  - Pull – Application iterates over XML entries

    http://developer.android.com/reference/org/xmlpull/v1/XmlPullParser.html

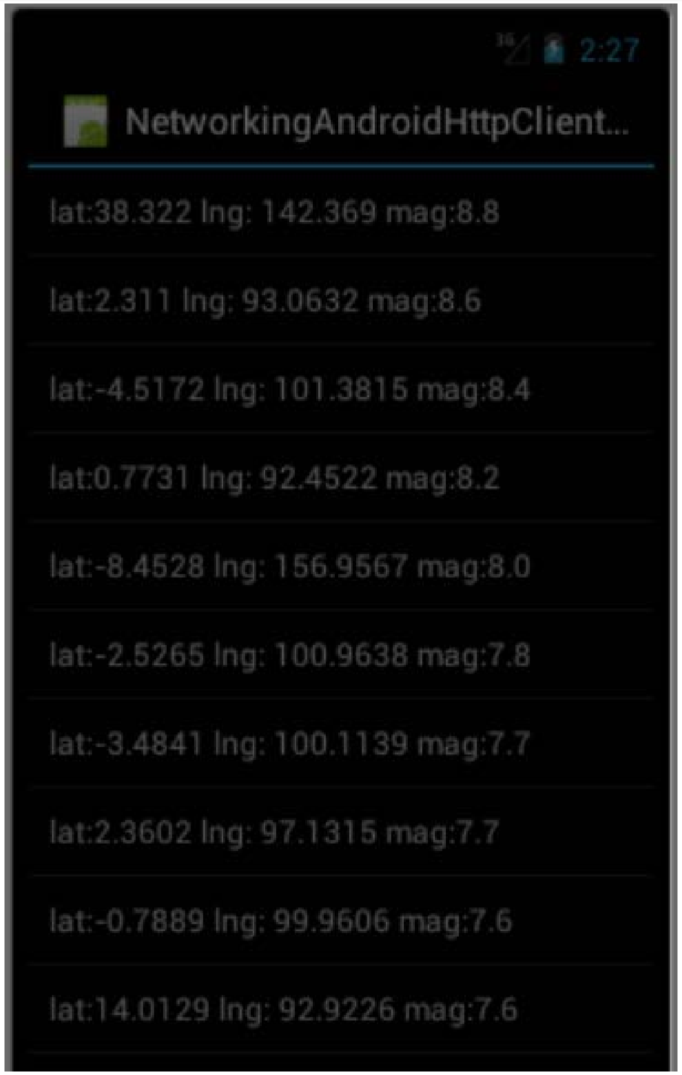




Earthquake Data (XML)

- http://api.geonames.org/earthquakes?north=44.1& south=-9.9&east=-22.4&west=55.2& username=demo
- Produces

```
<geonames>
    <earthquake>
            <src>us</src>
            <eqid>c0001xgp</eqid>
            <datetime>2011-03-11
                    04:46:23
            </datetime>
            <lat>38.322</lat>
            <lng>142.369</lng>
            <magnitude>8.8</magnitude>
            <depth>24.4</depth>
    </earthquake>
    ...
</geonames>
```



```
lat:38.322 lng: 142.369 mag:8.8
lat:2.311 lng: 93.0632 mag:8.6
lat:-4.5172 lng: 101.3815 mag:8.4
lat:0.7731 lng: 92.4522 mag:8.2
lat:-8.4528 lng: 156.9567 mag:8.0
lat:-2.5265 lng: 100.9638 mag:7.8
lat:-3.4841 lng: 100.1139 mag:7.7
lat:2.3602 lng: 97.1315 mag:7.7
lat:-0.7889 lng: 99.9606 mag:7.6
lat:14.0129 lng: 92.9226 mag:7.6
```

Parsing XML (Sax-Based)

```
…
private class XMLResponseHandler implements
                                                ResponseHandler<List<String>> {
  public List<String> handleResponse(HttpResponse response)
                                  throws ClientProtocolException, IOException {
    try {
      SAXParserFactory spf = SAXParserFactory.newInstance();
      SAXParser sp = spf.newSAXParser();
      XMLReader xr = sp.getXMLReader();
      XMLContentHandler handler = new XMLContentHandler();
      xr.setContentHandler(handler);
      xr.parse(new InputSource(response.getEntity().getContent()));
      return handler.getData();
    } catch …
    return null;
  }
}…
```

Parsing XML (Sax-Based)

```
private class XMLContentHandler extends DefaultHandler {
  String lat = null, lng = null, mag=null;
  boolean parsingLat = false, parsingLng = false, parsingMag=false;
  List<String> results = new ArrayList<String>();

  public void startElement(String uri, String localName,
          String qName,Attributes attributes) throws SAXException {
    if (localName.equals("lat"))
    {  parsingLat = true; }
    else if (localName.equals("lng"))
    { parsingLng = true; }
    else if (localName.equals("magnitude"))
    { parsingMag = true; }
}
```

Parsing XML (Sax-Based)

```
public void characters(char[] ch, int start, int length)
                                                    throws SAXException {
  if (parsingLat) {
    lat = new String(ch, start, length).trim();
  } else if …
}

public void endElement(String uri, String localName, String qName) {
  if (localName.equals("lat")) {
    parsingLat = false;
  } else if …
  } else if (localName.equals("earthquake")) {
    results.add("lat:" + lat + " lng: " + lng + " mag:" + mag);
    lat = null; lng = null; mag = null;
  }
}

public List<String> getData() { return results; }
…
```

Source Code Examples

- NetworkingSockets
- NetworkingURL
- NetworkingAndroidHttpClient
- NetworkingAndroidHttpClientJSON
- NetworkingAndroidHttpClientXML