



Hypertext Transfer Protocol (HTTP)

# Programming the Android Platform

CS 282

Principles of Operating Systems II  
Systems Programming for Android

# HTTP Overview

- The Hypertext Transfer Protocol (HTTP) is an application-level request/response protocol for sending web content
- Every web page that you visit, including Facebook, is retrieved using HTTP
- HTTP has expanded beyond transferring web pages & is now used as the basis for many other protocols, such as the Session Initiation Protocol (SIP) that is used for voice over IP (VOIP)



# HTTP To Get Your Facebook Feed

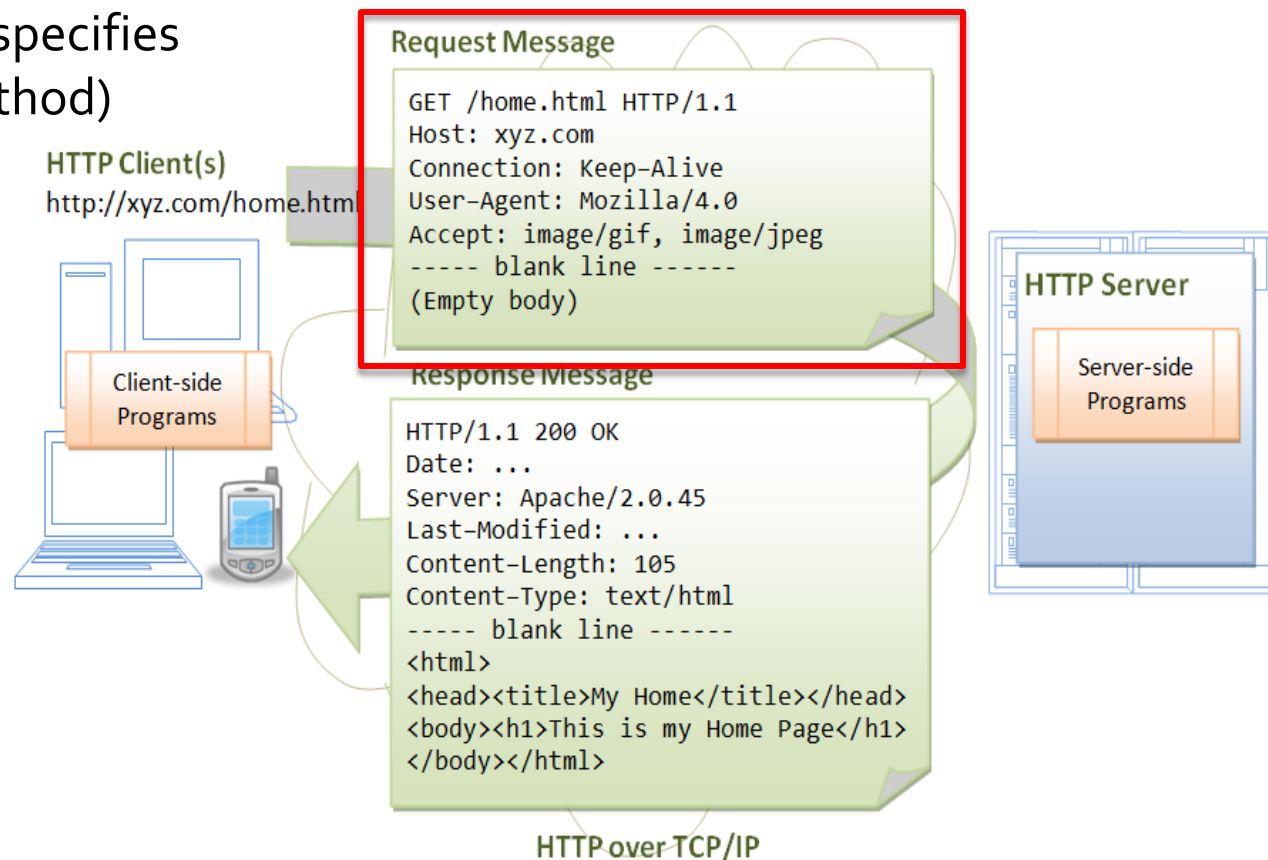


GET /home.php

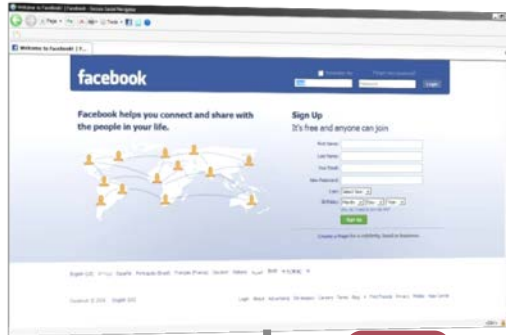
HTTP/1.1 200 OK, followed by your feed

# HTTP Request Message

- A request message is sent to the server to take an action on a resource located on the server
- Requests are formatted in three key parts:
  - First, the request line specifies an action (request method) & a target resource on the server
  - Next, a series of headers are attached that include metadata
  - Finally, an optional message body follows



# HTTP Request Method



GET /home.php


Request  
Method

HTTP/1.1 200 OK, followed by your feed

# HTTP Request Method

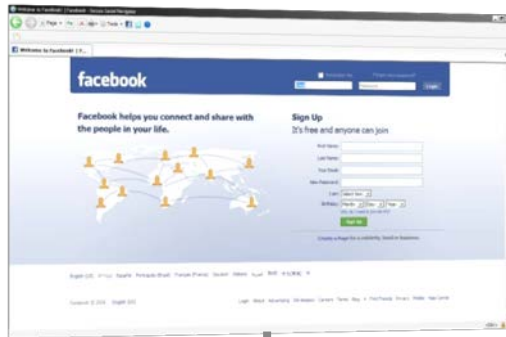
- The “request method” tells the server what action you want to be taken on the resource that you have indicated in your request
- The most basic/common request method is to GET a resource from the server
  - GET my feed & send it to me
- There are a variety of request methods:

- GET
- POST
- PUT
- HEAD
- DELETE
- TRACE
- OPTIONS
- CONNECT
- PATCH



These are by far the two most important & common request methods

# HTTP Resources



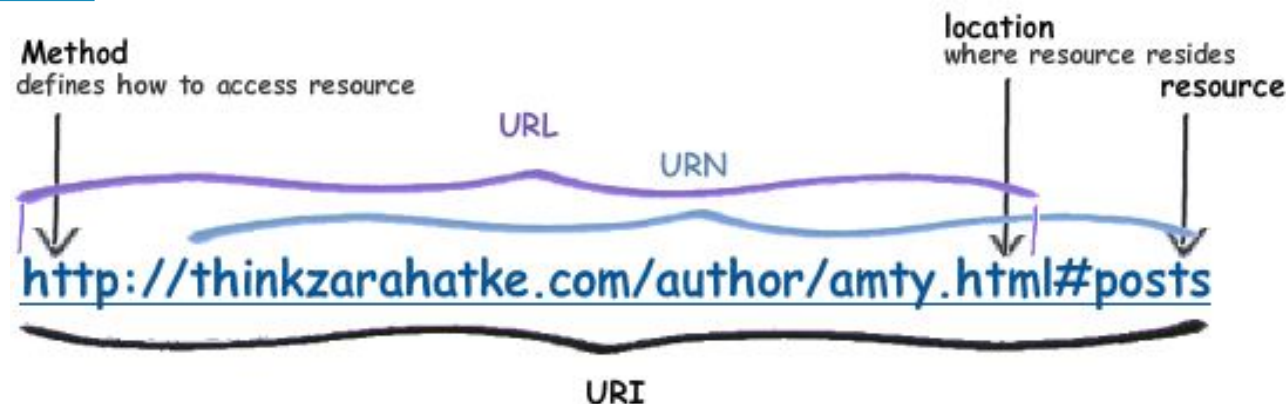
GET /home.php

Resource

HTTP/1.1 200 OK, followed by your feed

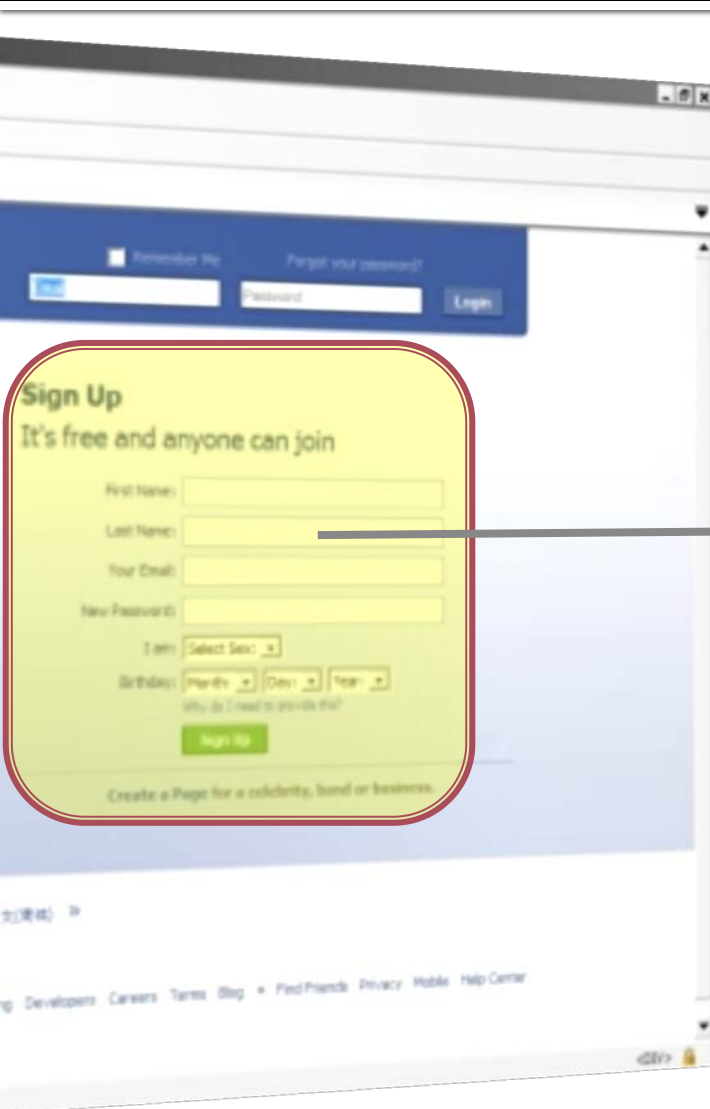
# Resources / URLs

- The resource is the \*thing\* on the server that you want the action to affect
  - e.g., “home.php” is the resource that has your Facebook feed
- Resources are referred to by Uniform Resource Locators (URLs)
- A URL is built of multiple parts: <scheme>://<server>/<resource>
  - e.g.: <http://www.facebook.com/home.php>
- The scheme is how to access the resource, <http://> means that the resource should be accessed using HTTP
- The server is the network location that hosts the resource
  - e.g.: [www.facebook.com](http://www.facebook.com)
- The resource is the thing that you want to act on
  - e.g.: home.php



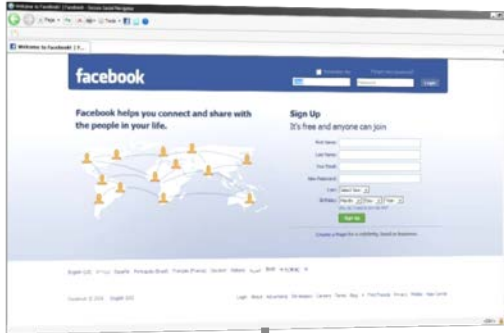


# Sending Data with Requests



When you sign up for Facebook, how does the data get sent to their server?

# HTTP POST



POST allows you to  
send extra data with  
the request



POST /signup.php + DATA

HTTP/1.1 200 OK, followed by terms and  
conditions to sign away your privacy

# HTTP POST

- A POST request can include parameters other than the resource to access
- These parameters are a series of Key/Value pairs that are sent to the server
  - e.g., FirstName=John LastName=Doe SignAwayPrivacy=True
- The data is sent in the body of the message
- Here's a small example:

POST /login.php?login\_attempt=1 HTTP/1.1

Accept:application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,\*/\*;q=0.5

Content-Type:application/x-www-form-urlencoded

Origin:http://www.facebook.com

Referer:http://www.facebook.com/

User-Agent:Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10\_6\_3; en-US)

AppleWebKit/533.4 (KHTML, like Gecko) Chrome/5.0.375.53 Safari/533.4

locale:en\_US

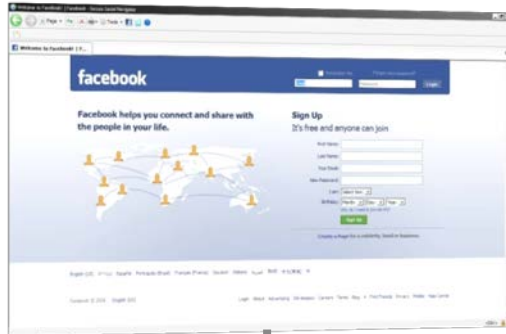
email:john.doe@gmail.com

pass:somepassword

# Can't You Send Data with GET?!?

- You can also send data with a GET request
- One key difference between sending data with GET & POST is that the data you send is encoded into the request URL
  - This limits the length & content of the data you can send
  - You aren't going to send inappropriate photos of friends to Facebook encoded as a URL (not that you should be sending those anyways)
- Example URL with parameters:
  - `http://www.facebook.com/signup?FirstName=John&LastName=Doe`
  - The start of the parameter set is marked by "?"
  - Each parameter is separated by "&"
- The parameters you send must follow the encoding rules for URLs (e.g. no spaces)

# HTTP Resources



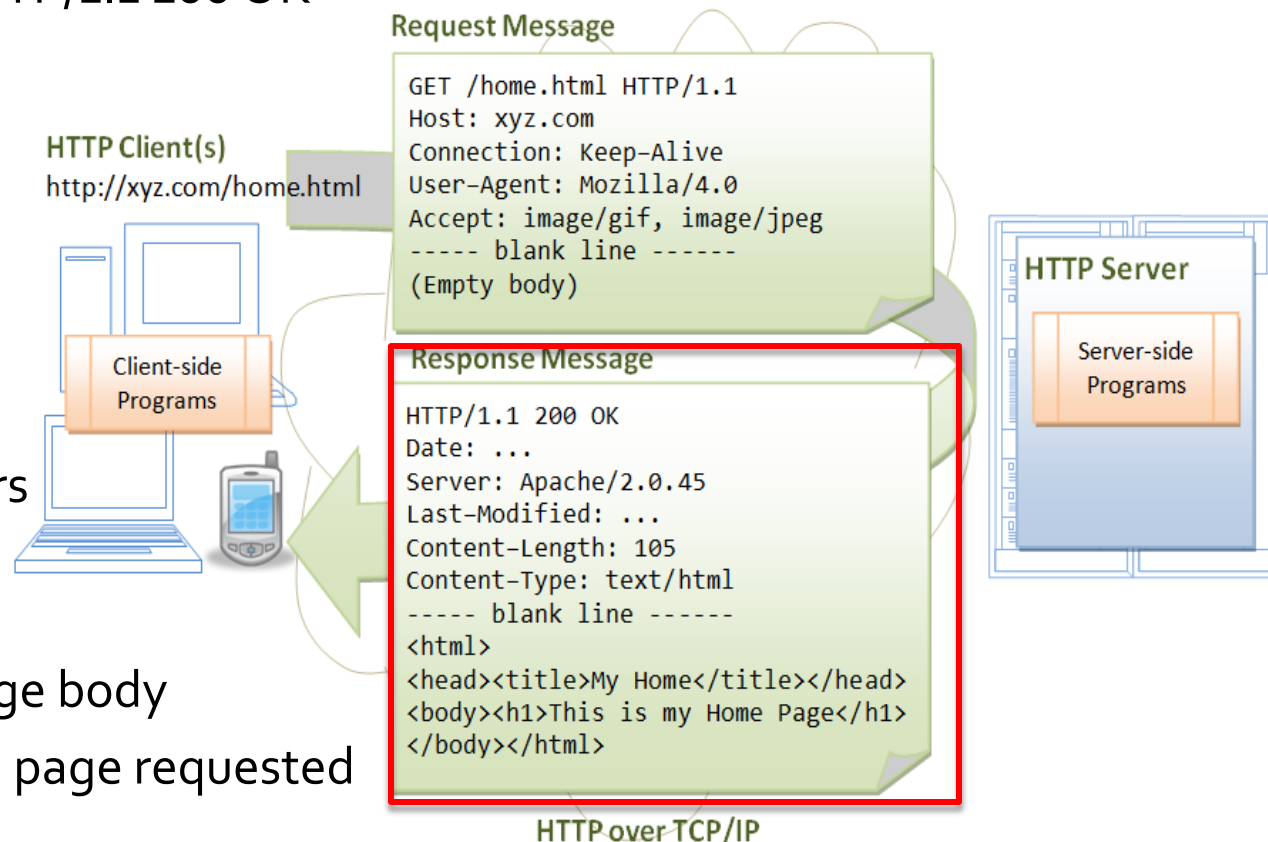
GET /home.php

HTTP/1.1 200 OK, followed by your feed

Response

# HTTP Response Message

- A response message is sent from the server to a client a result of a request
- Responses are formatted in three parts:
  - First, status line specifies what happened on server (e.g., was resource found)
    - e.g., (we found it): HTTP/1.1 200 OK
    - The status line starts with the version of HTTP being used
    - A status code
    - A readable version of the status code
  - Next, a series of headers are attached including metadata
  - Finally, optional message body
    - e.g., content of web page requested



# HTTP Status Codes

- 1xx – informational message
- 2xx - success
- 3xx – redirect somewhere else
- 4xx – you messed up (the client made an error)
- 5xx – the server messed up
- Common status codes:
  - 200 OK – you got lucky & everything was fine
  - 404 Not Found – You asked for a resource the server doesn't have
  - 500 Server Error – a script on the server blew up
  - 301 Moved Permanently – that resource moved elsewhere

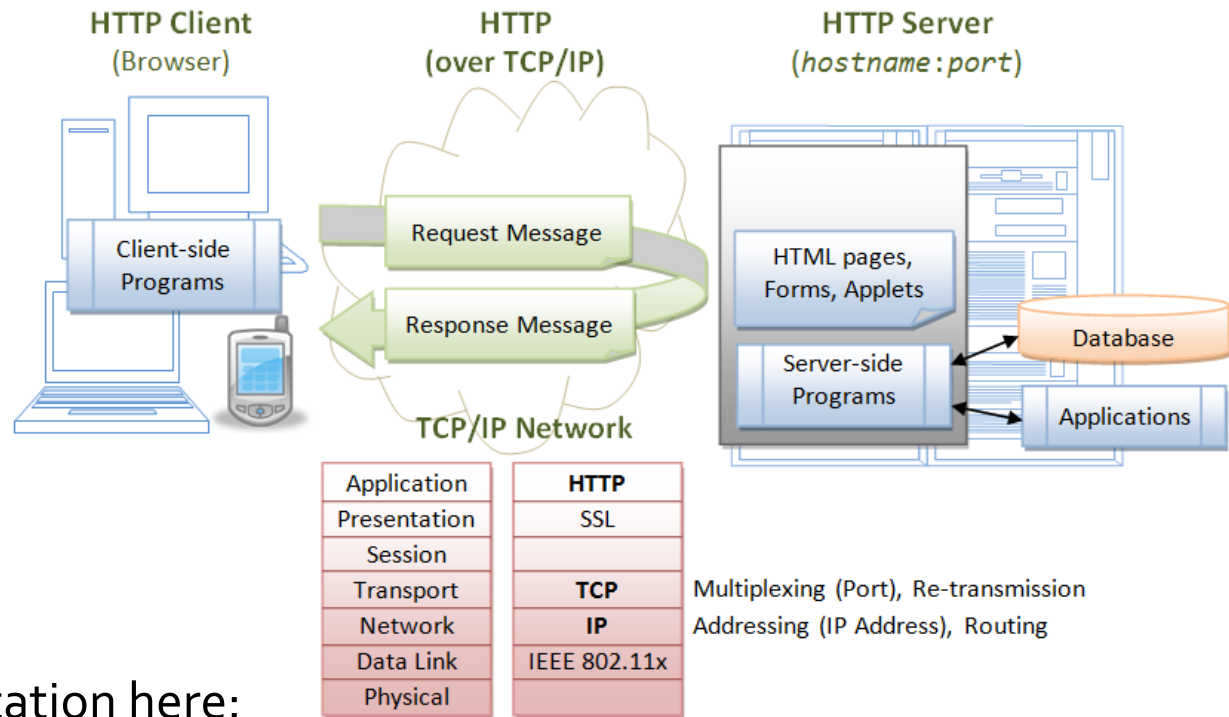
## HTTP STATUS CODES

<b>200</b>			
OK/Success	Everyone arrives at Page A. There is much rejoicing!		
<b>301</b>			
Permanent*	Everyone is redirected to the new location, Page B.		
<b>302</b>			
Temporary*	Visitors and bots are redirected. Juice is left behind.		
<b>404</b>			
Not Found	Original page is gone. Visitors may see a 404 page.		
<b>500</b>			
Server Error	No page is returned. Everyone is lost and confused :(		
<b>503</b>			
Unavailable	Asks everyone to come back later. A 404 alternative.		

\* Technically, code 301 is "Moved Permanently" and 302 is "Found", but SEOs refer to them as "Permanent Redirect" and "Temporary Redirect".

# Android HTTP Support

- Android has built-in HTTP support via the Apache HttpClient library
- You can also use Java's URL class to send an HTTP request & get the response
- The key classes are contained in org.apache.http.client



- See the documentation here:
  - <http://developer.android.com/reference/org/apache/http/client/package-summary.html>



# Sending an HTTP GET with Android

## ■ Example:

```
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;

public class HTTPRequestSender {
    public void getResource(String url){
        try {
            HttpClient client = new DefaultHttpClient();
            HttpGet request = new HttpGet(url);
            HttpResponse response = client.execute(request);
            //do something with the response, e.g., check the status code, get the entity, etc.
        } catch (Exception e) { /* that didn't work... */ }
    }
}
```

# Extracting the Response Body

## ■ Examples:

```
HTTPResponse response =
    client.execute(request);

StatusLine statusLine =
    response.getStatusLine();

if (200 != statusLine.getStatusCode())
    return;

HttpEntity entity = response.getEntity();

byte[] bytes =
    EntityUtils.toByteArray(entity);

//do something with the bytes
```

```
HTTPResponse response =
    client.execute(request);

StatusLine statusLine =
    response.getStatusLine();

if (200 != statusLine.getStatusCode())
    return;

InputStream in =
    response.getEntity().getContent();

byte[] data = new byte[1024];
int bytesread = 0;

while((bytesread = in.read(data)) != -1){
    //do something with the bytes
}
```

# What is an InputStream?

- An InputStream in Java is an object that you can read data from
- InputStream's allow you to read data into byte arrays:

```
...
InputStream in = response.getEntity().getContent();

byte[] data = new byte[1024];

//this call reads bytes from InputStream & puts them into the data byte array
int bytesread = in.read(data);

//the number of bytes that were read in the read call are returned from the method
//if -1 is returned, it means the InputStream doesn't have any more data
if(bytesread == -1){ /* ... */ }
else { /* ... */ }
//always close an InputStream when you are done
in.close();
```

# What is an InputStream? (cont'd)

- An InputStream is an abstract base class, you cannot instantiate it
- Common concrete types of InputStreams:
  - FileInputStream
  - ByteArrayInputStream
  - BufferedInputStream
- Examples:

```
//Read a file
FileInputStream fin =
    new FileInputStream("c:/somefile.txt");
//Turn a string into an InputStream
ByteArrayInputStream bin =
    new ByteArrayInputStream("foo".getBytes());
```

# Converting InputStream to String

- An InputStream can be turned into a String like this

```
import java.io.BufferedReader;
import java.io.InputStreamReader;

String line = null;
try {
    BufferedReader reader = new BufferedReader(new InputStreamReader(is));
    StringBuilder sb = new StringBuilder();

    while ((line = reader.readLine()) != null) {
        sb.append(line + "\n");
    }
} catch (IOException e) {
    e.printStackTrace();
}

String httpResponseval = sb.toString();
```

# OutputStream

- An OutputStream is an object that you use to write byte data to something
  - Example: write bytes to a file
  - Example: write bytes to the network
- Examples:

```
//Write to a file
FileOutputStream fout = new FileOutputStream("c:/somefile.txt");

String somedata = "This will be written to the file";
byte[] bytes = somedata.getBytes(); //bad form

fout.write(bytes); //Write the data

//Ensure that the data actually was written & isn't buffered in memory...if you don't
//do this...you will regret it one day...
fout.flush();

//Clean up after yourself
fout.close();
```

# Sending an HTTP POST w/Android

## ■ Example

```
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;

public class HTTPRequestSender {
    public void getResource(String url){
        try {
            HttpClient client = new DefaultHttpClient();
            HttpPost request = new HttpPost(url);
            List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(2);
            nameValuePairs.add(new BasicNameValuePair("name","some value"));
            nameValuePairs.add(new BasicNameValuePair("another name","another value"));
            UrlEncodedFormEntity entity = new UrlEncodedFormEntity(nameValuePairs);
            request.setEntity(entity);
            HttpResponse response = client.execute(request);
            //do something with the response
            ...
        }
    }
}
```