# HTTP URL Connection

The links provided in the topics area seemed more related to Android development, but since it is java based I don’t think it mattered. I’ve done HTTP URL connections in some PHP applications that I’ve developed in the past, so I’m familiar with the purpose behind them and why they are utilized. It is a simple method for pulling and pushing data over a standard HTTP/HTTPS connection. The side you are connecting to must have a valid listener for this to function appropriately as well as an application that understands HTML. But this is a great way to pull down information from repositories and to utilize it within your application. You can also push information to a listening Apache, IIS, or Tomcat instance so store information on a server.

In Vogella’s example (<http://www.vogella.com/tutorials/AndroidNetworking/article.html>) it provides a very simple Java HTTPURLConnection example on Step 3. The main class creates the URL resource record under the name “url.” An HttpURLConnection is then opened against that URL pointer and the stream is read. This is just like any normal file stream at this point, without any parsing of the HTML itself.

The example code can be found at the following URL: <https://github.com/hodges-olan/CIT360-Portfolio/tree/master/CIT360-Portfolio/src/HTTPURLConnection>

This method could be used to send JSON over an HTTP/HTTPS connection from my application client to a servlet where the centralized database resides. I will provide examples of that at a later date.

# Client & Server Sockets

In a network communication, the server will have an application that is waiting for users to connect to it. This is called a listener. A listener is combination of a protocol (either TCP or UDP) and a number (ranging from 1-65535). Only one application can listen on a port at a time, so your port needs to be available on the server. In this case, I’m going to only talk about TCP, as the majority of applications utilize TCP. The client will begin by sending out a TCP SYN packet to the server on the specified port, which is basically the client knocking on the servers door. The server will respond by opening the door and saying hi by replying with a TCP SYN ACK. And finally the client will respond with a hi in a TCP ACK message. They now can begin exchanging information with each other. Once complete, they will say bye (through a series of TCP FIN, FIN ACK, and ACK messages) and their transaction is complete. I don’t have a URL for this information, as I work with networks day in and day out as a network engineer.

Implementing this was another task though. I found the following URL which gave several examples of a server/client application. I chose to use the date application since it was the easiest to understand the socket establishment process without having lots of overhead for application coding. My code is at the following GitHub URL: <https://github.com/hodges-olan/CIT360-Portfolio/tree/master/CIT360-Portfolio/src/ClientAndServerSockets>

The place where I found the tutorial was here. <http://cs.lmu.edu/~ray/notes/javanetexamples/>

I also posted a video demonstrating the functionality at the following Youtube video.

<http://youtu.be/AySrKp0Bf1w>

# Activity Diagram

Activity diagrams are a method for the programmer to document how a program functions from start to end. It allows the developer to formulate how the application will flow from startup to shutdown and what input/output it will require. This sounds similar to the sequence diagram, but this is a higher level diagram. Sequence diagrams are at the object level, where activity diagrams are at the functional level. You will not see the flow of messages/calls between the different objects, nor will you see the lifetime of these objects. But it will show you what the user would experience step by step through the application. I was able to brush up on this by reading at Tutorialspoint.com: <http://www.tutorialspoint.com/uml/uml_activity_diagram.htm>

I had previously done two activity diagrams in the CIT 262 class. The link to those can be found here:

<https://drive.google.com/open?id=0B5_UhQZPuf-5M0pxZ1R6V1VJc2c>

I plan on creating an activity diagram for my personal application which I will submit at a later date.

# Application Controller Pattern

From my research this week, I was able to find out that the Application Controller Pattern is all about centralized request handling. From what I was able to understand, I feel that this mostly works well when performing authentication/authorization/accounting (AAA) of all application activities. Having a centralized process to manage AAA gives you ease of control over what each user can do, and what they can view.

Tutorialspoint.com referenced this pattern as the Front Controller Pattern. The main class interacts directly with the front controller. In turn, the front controller places calls out to the specific view and backend controller methods via a dispatcher class. <http://www.tutorialspoint.com/design_pattern/front_controller_pattern.htm>

I also caught a video of a group of BYUI students that were discussing this pattern at length at the following URL: <https://www.youtube.com/watch?v=BpoQGC0SQoI> Their application was a simple calculator application which utilized an application controller implementing with a HashMap of all the available commands and associated controller methods to be utilized with those commands.

I duplicated their application to verify it’s functionality at the following GitHub URL:

<https://github.com/hodges-olan/CIT360-Portfolio/tree/master/CIT360-Portfolio/src/ApplicationControllerPattern>

I plan on utilizing this centralized controller method with my personal application to provide the two separate views that will be needed; the member view and the admin view. This will allow me to authenticate the user before allowing them into the application, and once authenticated I can authorize them to a specific view within the application.

# JSON Serialization

I’m sorry if I didn’t make this clear last time I submitted this topic, but the Vehicles example I gave was a creation of my own. I felt it was the easiest way to show how JSON serialization works as vehicles can have multiple attributes in and of themselves. But I did modify the application with a client and server side option now. The client creates the objects and serializes them. It then transfers the data to the server, which in turn saves the information to a text file. It can then ask for the data back and rebuild the object through this same data transfer process. This can be found at the following GitHub URL: <https://github.com/hodges-olan/CIT360-Portfolio/tree/master/CIT360-Portfolio/src/JSONSerialization>

The file names are VehicleClient.class and VehicleServer.class

# Model View Controller Pattern

I’ve gone ahead and put a different application in this topic as I did not receive the application from my team mate. I was able to find this example of a student and his associated data. The main application creates instances of all three (model, view, controller) and it assigns the model and view to the controller it created. From there on the main application performs actions against the controller class only to display data, update data, etc. The example can be found at the following GitHub URL:

<https://github.com/hodges-olan/CIT360-Portfolio/tree/master/CIT360-Portfolio/src/ModelViewControllerPattern>