CSE 310 Interest Groups Project  
by Steven Herring and James Morris

**Overview**

Our project provides the following functionality:

1. Help

2. Login

3. RG

3. AG

4. SG

1. The *Help* function provides the user with a brief description of each function currently provided, including the parameters for operation.

2. The *Login* feature allows the user to login and maintain a preserved state of activity on the forum.

3. The *RG* feature allows the user to browse a certain group’s posts and read/write to the group.

3. The *AG* feature lists all the groups available and specifies whether the user is subscribed or not.

3. The *SG* feature lists all the groups that the user is subscribed to, the functionality also lets the user know of the potential new messages.

In all instances, the server holds a cache of important data regarding both user accounts and group information. However, the client keeps its own instance to which groups it is subscribed.

**User Documentation**

Our group used a standard Client/Server solution to the presented assignment. Therefore, both programs must be executed for intended results.

The project was coded and tested with python 3.x  
  
**Interest Groups Server:**

The server must be running on a known host. Obtain the IP from that machine using a command such as ifconfig. The server may be run with a debug flag (“-d”), to help give more information during runtime. To start the server, navigate to the root directory of the source code bundle and execute…

python3 server/discussionServer.py

This will start a server, using the pre-existing data set given, data created throughout previous executions of the server and interactions with clients.

**Interest Groups Client:**

Clients must be privy to the IP address of the host machine. The client must also know about their own ID number. For testing purposes, you can use any of the existing users…

1337

1111

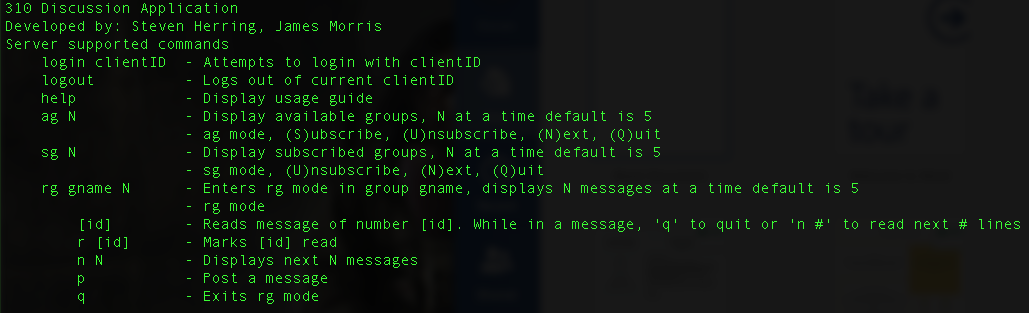
2222

3333

4444

For the client to connect to connect to the server, run the following command from the root of the source code bundle…

python3 client/discussionServer.py [IP\_ADDRESS]

After connection the client must type ‘login [ID]’ to login. And may use the ‘help’ command to get the most up to date version of the following…  
  


**System Documentation**

**Overview:**

The client and server work together to produce desired results, but the server does most of the heavy lifting. The client will pull and cache any requested data from the server, and perform sanity checks on future requests to confirm it has matching data with the server. The client machine stores data specific to the client, read posts, subscriptions, things like that. Though the server does maintain copies of this information for those times the client uses different machines to login.

**Data Transfer Protocols:**

We use a TCP socket stream to transfer data back and forth between client and server. There is a separate module created for this task named “da\_protocols.py”. From this module, we have two functions

* senddata(socket, data, packetlength, packetdelim)
* receivedata(socket, data, packetlength, packetdelim)

Senddata takes in a JSON valid dictionary, decomposes it into a JSON string, splits it into smaller sized chunks if it’s size is larger than our maximum packet size. Then it sends a packet identifying how many packets the application should expect and the rest of data behind it through the socket.

Receivedata takes in the first packet, identifying how many packets it will be received. Then stitches them back together, after receiving all packets it restructures the JSON string as a dictionary and returns the completed dictionary.

The idea here is JSON in, JSON out. All the dirty work gets done internally.

**Concurrent Operations:**

Every operation that reads or writes server data is done atomically. A client thread must acquire a lock on the data, read or write the data then release the lock so the next client can read or write.

**Data Stores:**

Data is stored, sent and received using JavaScript Object Notation (JSON). This gives a very easy and modifiable data structure set up, allowing developers to add any data they need to be saved or transferred very easily.

On global data updates (group content, client accounts), the server saves all of its data into associated \*.json files.

* Client Data: server/clients/ids.json
* Group Data: server/groups/groups.json
* Group Content: serve/groups/threads/\*.json

The server loads and caches all data globally upon start, and throughout its lifetime updates the caches then writes them out to the pairing file prior to termination. Using a lock all data caches and files can only be read from or written by one client thread at any given time, preventing loss of data.

Important data is marked with a modification time. The client also caches data it pulls from the server. Before ever displaying data to the user, the client confirms its modification time matches the server, if it’s a match it doesn’t request the update from the server. In the case modification times are different, the server sends the client new data to replace the old.

**Data Structures:**

The most important data structures for the server are the global ones that all threads share access to. These structures handle all important information that we send to client and want them to be in sync. These are all dictionaries, that maintain a structure of important information.

* clients – cache for clients and their login status
* groups – cache for group names and their data files
* groupsContent – cache for subject threads, posts, etc.

**Program Limitations:**

* Maximum Clients – No known limit
* Maximum Post Size – No limit

**Testing Documentation**

The server comes prepackaged with data for testing convenience. This data includes users, groups, posts in the groups. Because the data is preserved, across multiple runs we some scenarios may look different than described below. The below results will be identical if tested on a freshly extracted version of the software.

**Testing routine:** From the server machine, execute the following:

Python3 server/discussionServer.py

From the client machine, execute the following:

Python3 client/discussionClient.py [IP ADDRESS]

The testing routine format will look like this…

1. *First Command To Enter*
   1. *Expected Result*
2. *Second command*
   1. *Expected Result*

Continuing from the client process use the following commands.

**1-5** Demonstrates the ability to login, logout, log back in and quit the program.  
**6-12** Demonstrates AG mode, all its functionality and ability to leave AG mode.  
**13-15** Demonstrates SG mode, all its functionality and ability to leave SG mode.  
**16-22** Demonstrates RG mode, mark post read, create post, q

1. Help
   1. Displays the server usage menu.
2. Login 1337
   1. User 1337 has successfully logged in.
3. Logout
   1. User 1337 has successfully logged out.
4. Login 1111
   1. User 1111 has successfully logged in.
5. Quit OR q
   1. Goodbye! (Disconnects)
6. Python3 client/discussionClient.py [IP ADDRESS]
   1. Successful Connection!
7. Login 1337
   1. User 1337 has successfully logged in.
8. Ag n 3
   1. Displays 3 groups.
9. N
   1. Displays 2 more groups. (5 total)
10. U [#s ie. 1 2]
    1. Unsubscribes from groups, redisplays.
11. S [#s ie. 2 4]
    1. Subscribes to groups, redisplays.
12. Q
    1. Exits AG mode
13. Sg
    1. Lists groups subscribed & new posts since last viewed
14. U 4
    1. Unsubscribes from the 4th group
15. Q
    1. Exits SG mode
16. Rg group.2.com
    1. Enter RG mode, displays posts in group.2.com
17. R 1
    1. Marks the first message as read, removing the ‘N’ next to it
18. P
    1. >Subject:
19. Subject1
    1. >Author:
20. John Smith
    1. >Message:
21. Hey this is a post
    1. Adds a post to subject1 in group.2.com by John Smith saying “Hey this is a post”
22. Q
    1. Exits RG mode