# COMP3331 - Networks Assignment: Networking eDocuments (Report)

**Program Design**

* Reader
  + The reader's Book class updates itself (ie determines whether there are new messages) by using the Reader's Database class - requesting to send any unread messages, and sending what messages are read

**How System Works**

* Both reader and server have different directories, to support the ability for independent databases (which may have conflicting names) to be used
* Reader
  + Has a directory for each book, each containing the book's pages and contents. It will also load these books into memory, for faster access when it comes to accessing forum posts
  + Has a file 'BookListInfo' containing information about each book
  + The first message (containing data) sent to the server will be the reader's username, mode (push/pull) . This message has a particular format, described in 'Message Design'.
* Server
  + Does not keep the books - only keeps a database containing forum posts, as well as client information. The downside is that the server would not be able to check the validity of the posts (eg erroneous book name /page number / line number).
  + *Server Database*  
    Similar to the reader database, only there is no 'read status' stored for each post (as this is determined locally at the reader).

**Message Design**

The delimiter between parameters in a message is the '#' character, chosen as it is an uncommon character to include in normal messages (eg post content)

* Reader
  + Introduction message: used to provide the username and mode to the reader  
    '#Intro# [Username]# [Opmode]'
  + Exit message: Client sends a clean exit message to terminate its connection with server  
    '#Exit#[Username]'
  + Forum post: There are two messages, one for the post information, and one for the post content. This was because the post content could be long, and may (in the future) involve formatting characters, hence being processed separately.   
    Post info: '#NewPostInfo#SenderName#BookName#PageNumber#LineNumber'  
    Post content: '#NewPostContent#Content'

**Design Considerations and Tradeoffs**

* Databases (for reader and server)
  + Initially, the database was meant to be represented as multiple files stored on local disk, which would be read by a database class in the code. This would support the fact that forum posts will be preserved even when server is turned off. However, this would mean every time a query is made to retrieve database information, it would need to read the file repeatedly, as well as parsing the information according to a specific format. This is a lengthy process, and makes the code more complicated. Given the specification that the databases can be assumed to be initially empty, I changed the databases to simply be in memory - for faster access, and easier readability.

**Possible Improvements**

* Server Database
  + By convention of databases, 'BookName' can be replaced by an integer ID
* Books (and their information) could be stored on server side and downloaded to the client side
* Reader
  + Use a list instead of tuples to store information about forum posts- they're in a certain format, hence a list would prove faster access, and better readability if indexes were defined

**Issues**

* [Formatting issue] Reader.py: When the reader's listening thread prints to screen, it will print on the same line as the '>' character for the user input.
* [Execution issue] Reader.py: Any error that causes 'main' to stop execution (besides from 'Keyboard Interrupt') will cause the listening thread to continue running.

**Code References**

* Code used as a framework for creating sockets, and making connections between a server and client taken from "Socket Programming HOWTO" (https://docs.python.org/2/howto/sockets.html)
* Code used for handling multiple clients using 'Select' for networks was adapted from "Python Network Programming: Echo Server with Select"  
  (http://ilab.cs.byu.edu/python/select/echoserver.html)
* The template code for constructing and running threads was adapted from "Python Multithreaded Programming"  
  (http://www.tutorialspoint.com/python/python\_multithreading.htm)
* The code used to terminate a thread when main exits through a 'Keyboard Interrupt' exception. Adapted from "Closing all threads with a keyboard interrupt"  
  (http://stackoverflow.com/questions/11436502/closing-all-threads-with-a-keyboard-interrupt)