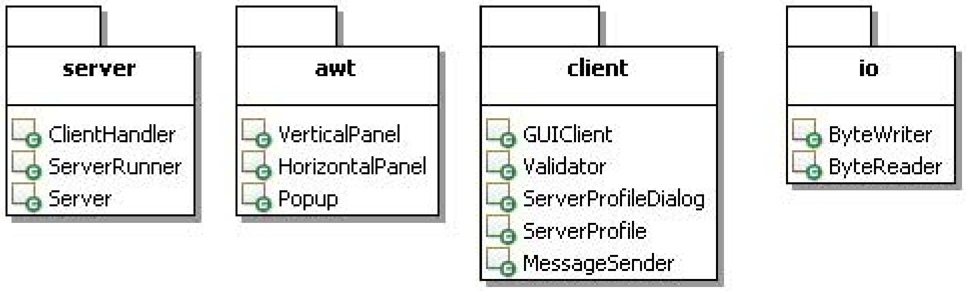
Practical 3 - TWIt

Introduction

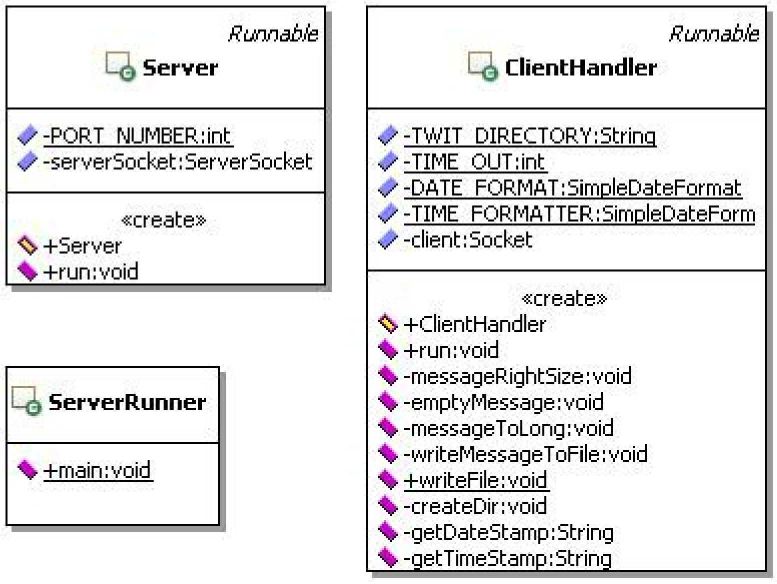
This practical requires a Java implementation of a TCP client/server which send/receive messages of up to 140 characters. This message, sent from a client to a server should be saved in directory on the server. From this location (public_html/twit/), the information can be accessed and displayed on a website.

UML: Class diagram

twit-packages:

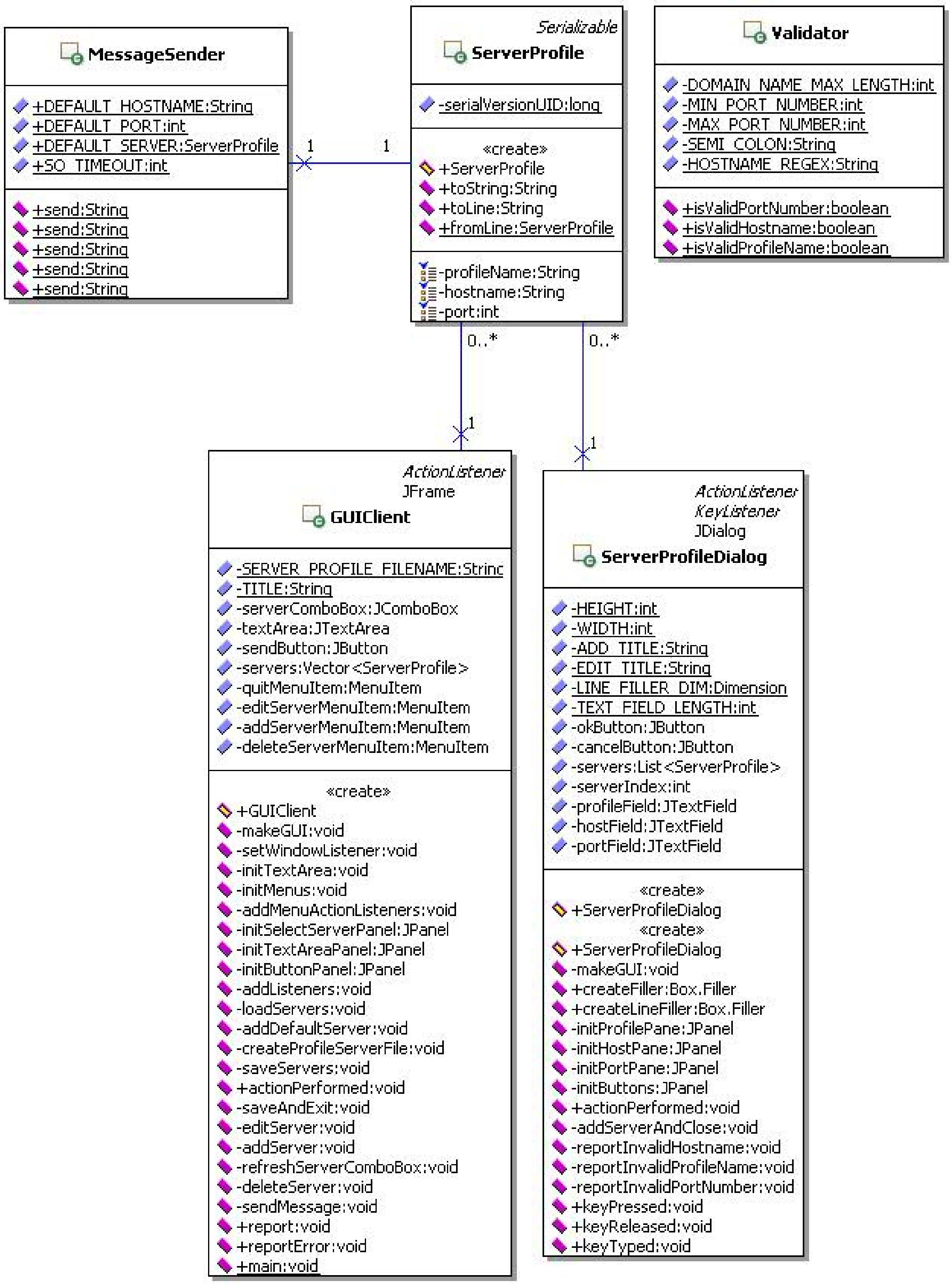


twit.server

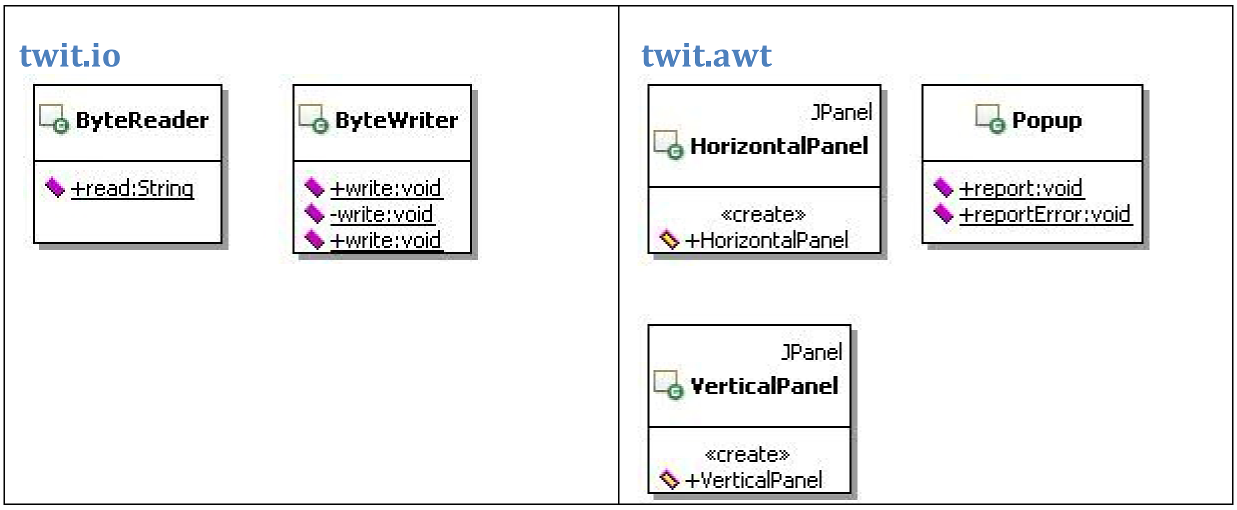


Server and ClientHandler both implement Runnable in order for the server to be multithreaded. This means that the server can handle several clients at the same time and can accept console input at the same time as accepting new clients.

twit.client



- ServerProfile implements Serializable. This has no purpose other than that of semantics and means that it can be converted to a String and back.
- Both GUIClient and ServerProfileDialog implement ActionListener. This means that the menu items and buttons are bound to a method which is called when something happens.

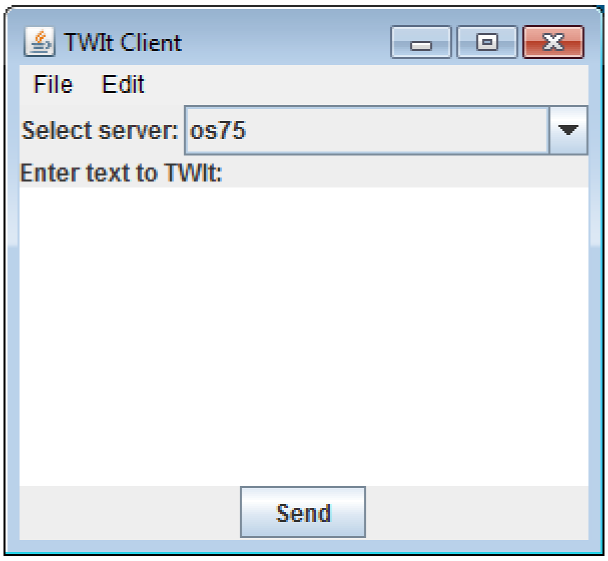


- HorizontalPanel and VerticalPanel are used to organise the layout of the GUI windows.

Client: Sending

The purpose of the client is to connect to a server and send a message. The client (found in twit/client) was implemented with a graphical user interface (GUI), which allows the user to manage a number of favourite servers and send messages to them. It can be found in twit/client/GUIClient.java. GUI applications are useful because they are more user-friendly and look nicer than console-based applications.

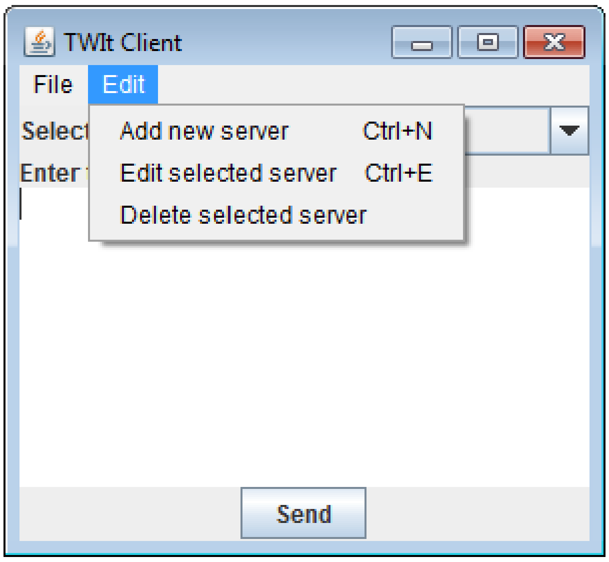
Sample output



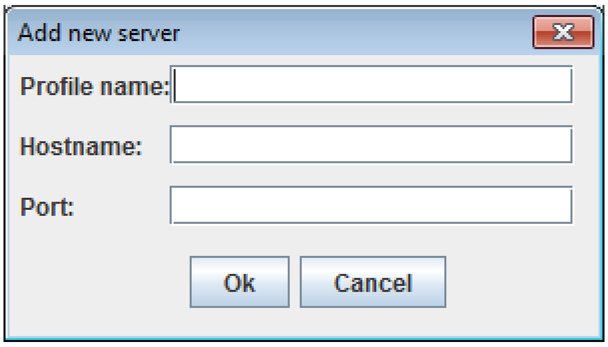
The main interface has two menus, a ComboBox to choose which of the favourite servers to send a message to, a TextArea in which to enter a message, and a send button. The majority of these elements are implemented using java.swing components.

The file menu only contains one option (or MenuItem): Quit. This MenuItem has a command associated with is (Ctrl+Q).

The edit menu (see picture to the right) contains options for changing the server profiles.

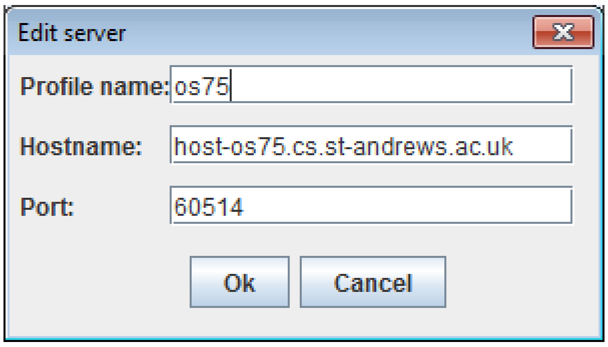


The "Add new server"-option opens the window below:



The user can input a profile name, hostname and port of the server they wish to connect to.

The "Edit selected server"-option opens a dialog similar to the one below:



Profile name: os75

Hostname: host-os75.cs.st-andrews.ac.uk

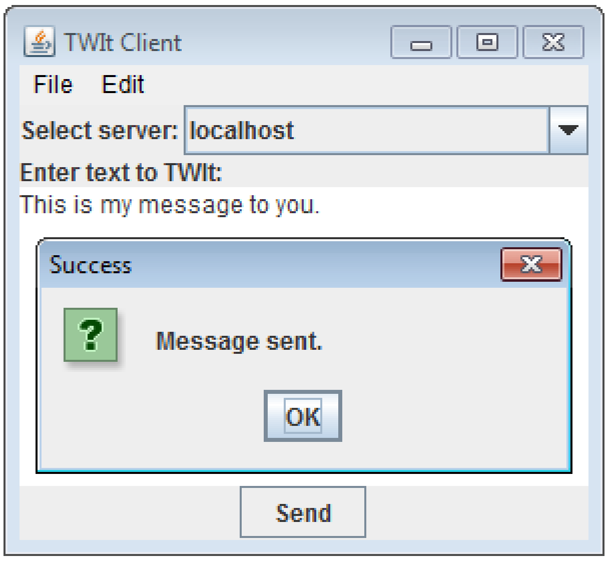
Port: 60514

The user can edit any information about the server. And click Ok. Any changes to the server-profiles are automatically saved while the application is being closed.

Validation of each of the fields is done when pressing the Ok button. This is done using twit.client.Validator.

Testing of TWIt Client

In the ideal scenario, where the message is of valid length (1-140 characters), sent successfully to the server and confirmation from the server is received, the following message is displayed:



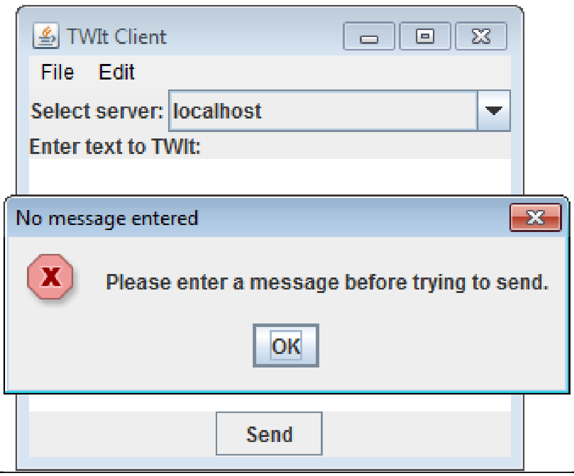
Select server: localhost

Enter text to TWIt:

This is my message to you.

Message sent.

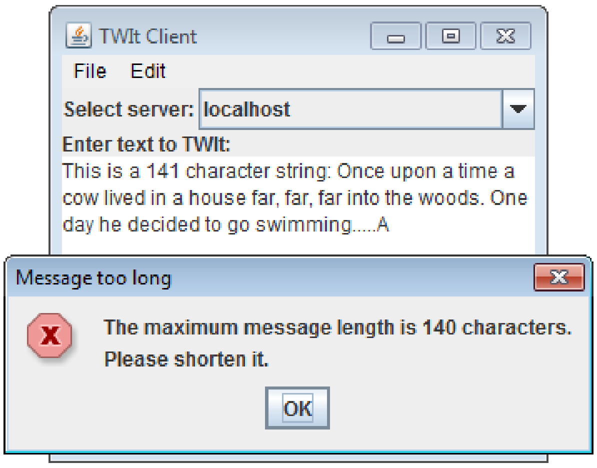
If, upon sending, no message has been entered, or the message entered only contains whitespace, the message below to the left is displayed to the user:



No message entered

Please enter a message before trying to send.

OK



This is a 141 character string: Once upon a time a cow lived in a house far, far, far into the woods. One day he decided to go swimming.....A

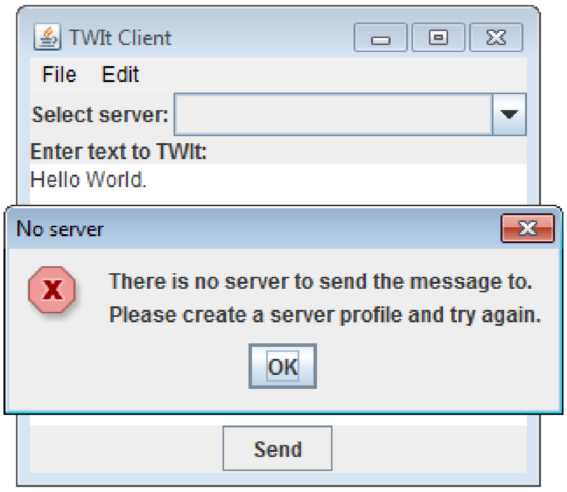
Message too long

The maximum message length is 140 characters. Please shorten it.

OK

If the message entered is too long (i.e. more than 140 characters), the message above to the right is returned.

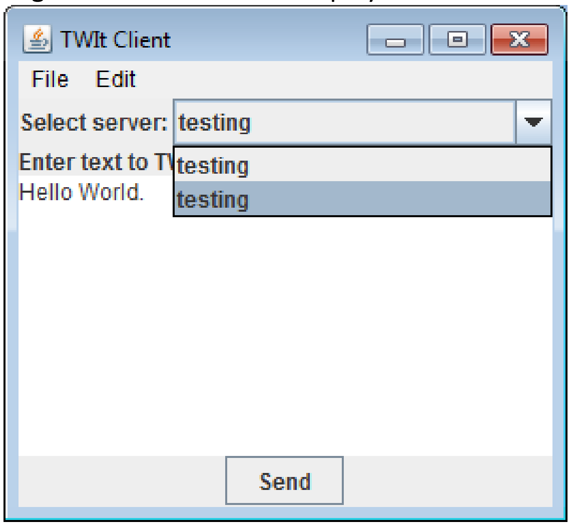
If there are no profile servers to choose the message below to the left is displayed:



No server

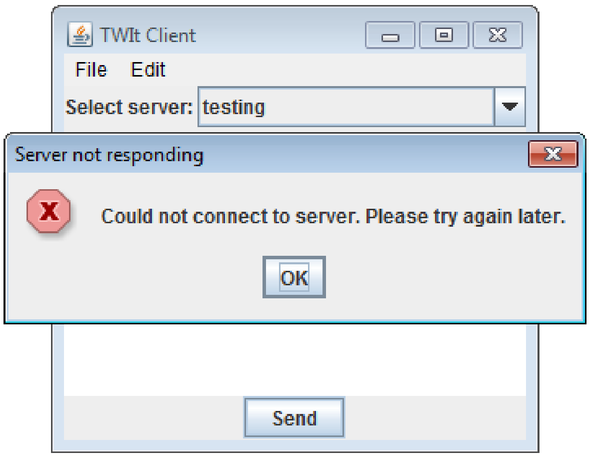
There is no server to send the message to. Please create a server profile and try again.

OK



Because of the limited time for the project, there is no mechanism for checking that two servers do not have the same profile name (as seen above to the right).

If, when attempting to send a message, the client cannot connect to the given hostname, the following message is displayed:



Server not responding

Could not connect to server. Please try again later.

OK

More exception handling

All of the following exceptional circumstances are dealt with and reported to the user:

- Problems reading from/writing to the server-profile-file.
- Connection-timeout. This happens after 15 seconds.

Other user errors

Error	Examples and method (if appropriate)
Trying to delete/edit profile-servers when none have been added	
Entering invalid input into the edit/new server profile dialog	Invalid profile name (profile names cannot contain semi-colons, because they are used to delimit the information in the server-profile file) Invalid hostname – this is dealt with using a regular expression* Invalid port number type (i.e. not an integer) Port number not falling within the given range

The regular expression (\\A[A-Za-z0-9]{1}[A-Za-z0-9-.][A-Za-z0-9]{1}\\z|\\A[A-Za-z0-9]{1}\\z) checks that the hostname begins and ends with a character or number, and otherwise only contains characters, numbers, dashes and full stops. It is therefore not perfect, because two dots cannot be right after each other. Additionally, internet has developed to allow certain international characters. twit/testing/HostnameValidationTesting tests this regular expression. An important thing to note is that it does not require dots to be present. This is because, locally, hostnames such as "host-os75" are allowed, and do work.

Server: Receiving and Saving

The server (found in twit/server) is used for receiving messages and saving them to the appropriate directory (public_html/twit/*date of posting*). The implementation of it is text-based, because the user of the server is expected to have more experience using computers than the user of the client. The server is multi-threaded, and can therefore deal with several clients at the same time, and accept user input into the server's console while receiving information from clients. There is currently only one command ("quit"), but more could easily be added in the future if needed. Under normal circumstances, i.e. after receiving an appropriately sized message (1-140 characters), the server will return the message to the client as a confirmation that it has been received.

Sample output

Here is some sample output of the server running, receiving a message which is too long, and a few appropriately sized ones. Testing was done with zero-length messages, but this had to be dealt with on client side rather than server-side, as it is not possible to send or receive empty messages.

```
C:\Users\Ole\Documents\Computer Science\Workspace\CS1004-P3-TWIt\bin>java
twit/server/ServerRunner
Starting server... Enter QUIT to quit.
Server socket established at Ole-Laptop:60514.
>> Connection established with serial.alcohol-soft.com:58486.
Message recevied is too long.
Connection closed.
Connection established with serial.alcohol-soft.com:58489.
Message written to file:
```

```
Hello World.
Connection closed.
Connection established with serial.alcohol-soft.com:58490.
Message written to file:
This is my message to you.
Connection closed.
Connection established with serial.alcohol-soft.com:58492.
Message written to file:
Today was fantastic.
Connection closed.
Connection established with serial.alcohol-soft.com:58493.
Message written to file:
This is a 140 character string: Once upon a time a cow lived in a house far, far, far into the woods.
One day he decided to go swimming.....
Connection closed.
quit
Shutting down server...
C:\Users\Ole\Documents\Computer Science\Workspace\CS1004-P3-TWIt\bin>
```

Ideally, one might want to know which client connection has been closed, but this is currently not something which the server gives information about.

Running the server

In order to make the server easier to run while testing, a very basic shell script was made. It is now located in the file "/cs/home/os75/server" and contains:

```
#!/bin/bash
cd eclipse/CS1004-P3/bin/
java twit/server/ServerRunner
```

All it does is to change the directory and run the Server, but it made testing of the server easier. It can be run by logging onto the server using SSH and using the command "sh server".

Error handling

The server checks if the message is too long by attempting to read 141 bytes from the connected clients' InputStream:

```
byte[] bytes = new byte[141];
// Read up to 141 bytes from inputStream into the array "bytes".
int length = inputStream.read(bytes);
// length now stores the number of bytes read into the array.
if (length > 140) {
	messageToLong(outputStream);
} else {
	messageRightSize(outputStream, bytes, length);
}
```

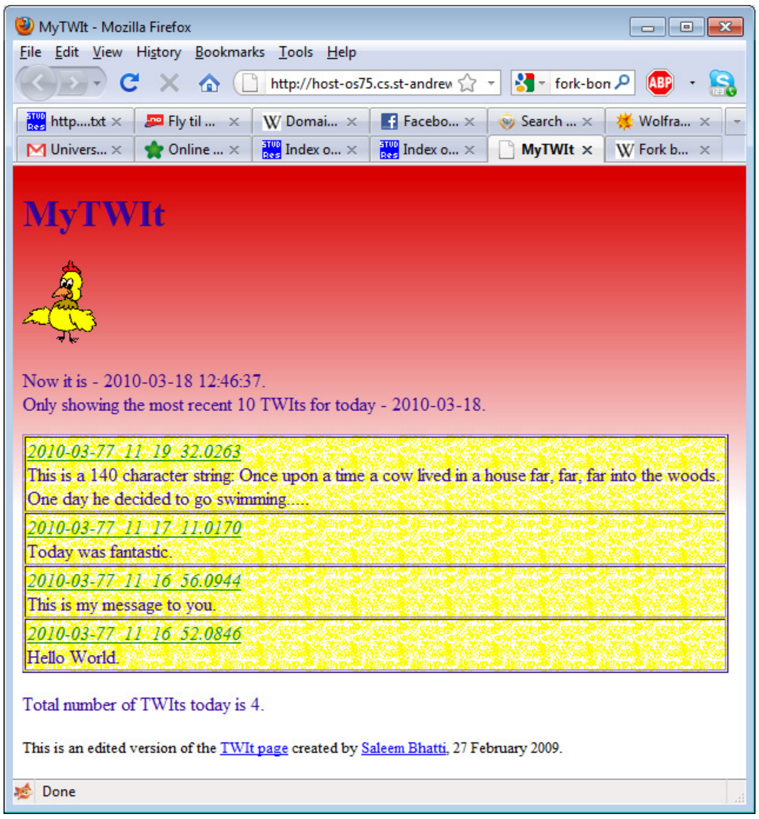
If the server receives more than 140 characters, it will return an error message to the client, instead of the usual confirmation. The purpose of this is not to deal with the GUIClient (which has length checking), but to deal with possible other clients which may attempt to connect. This method is testing in twit/testing/MessageTooLong.

Security issue

In the current implementation there is no mechanism checking the content of each message. This means that unpleasant JavaScripts can be received and run in the browsers of people accessing the webpage. This can obviously be a cause of problems. Examples include scripts which forward users to other sites and fork-bomb attacks.

Website: Displaying

The website displays the TWIts which have been posted on the same day the website is accessed. The current version of the page looks like this:



MyTWIt

Now it is - 2010-03-18 12:46:37.
Only showing the most recent 10 TWIts for today - 2010-03-18.

2010-03-77 11 19 32.0263
This is a 140 character string: Once upon a time a cow lived in a house far, far, far into the woods. One day he decided to go swimming.....

2010-03-77 11 17 11.0170
Today was fantastic.

2010-03-77 11 16 56.0944
This is my message to you.

2010-03-77 11 16 52.0846
Hello World.

Total number of TWIts today is 4.

This is an edited version of the TWIt page created by Saleem Bhatti, 27 February 2009.

The original was written by Saleem Bhatt on the 27th of February 2009. However, some modifications were made: It was made into valid XHTML 1.1, a fatal PHP error was fixed (see below), and it was connected to a CSS stylesheet.

Testing

The given index.php file in twit_html.zip reported a fatal error when there had not been any messages posted on the current day. This was fixed.

Validation link:
http://validator.w3.org/check?uri=http%3A%2F%2Fwww.cs.st-andrews.ac.uk%2F~os75%2Ftwit%2F

Use of example code

Although example code was given for many parts of the program, I chose to try to avoid using these as much as possible, and based my source code on information from lectures and Sun's JavaDocs and tutorials. I did, however, get inspired by the ByteWriter and ByteReader classes and I found out about the Socket.setSoTimeout() method from looking at TcpServer1c. I took the majority of the code for the date/time-stamps and creation of directories from the example given (https://studres.cs.st-andrews.ac.uk/CS1004-IP/Practicals/CS1004-twit/cs1004-examples.zip/cs1004-examples/DirAndFile.java). I also used the given index.php file as a basis for my website.

Conclusion

In this practical I had to write a client and server. The client would send messages to the server which would then save it to a file in a folder from which it could be displayed on a website. It was very successful in that it gave me an incentive to learn more about threads, socket programming, GUIs in Java (javax.swing/java.awt), (X)HTML, CSS, PHP, regular expressions, the UNIX shell and shell scripting. PHP and the UNIX shell were the only aspects in which I had no experience. However, the most difficult parts were creating a regular expression to match valid hostnames and setting up the GUI. In the case of the GUI I ended up having to use fillers to make it look as I desired.

Limitations to the implementation of the client are that international hostnames cannot be used (such as those containing the Norwegian characters 'æ', 'ø' and 'å') and lack of a checking mechanism for whether or not a new server profile has the same name as an existing one. Although a GUI client is nice, an online alternative would be preferable. This could have been done using CGI together with Java.

The limitations of the server are not as evident, but since it is written in Java it may be unable to deal with large number of clients. The majority of testing was done using a UNIX file system, and since the server saves incoming messages to an absolute UNIX path, it is unlikely that the server will be fully functional in a Windows environment. Also, as previously mentioned, it has little/no protection against attackers. Attackers may send JavaScript messages which can be harmful to visitors of the webpage, although these scripts are limited to 140 characters.