RIT | Golisano College of Computing and Information Sciences
School of Information

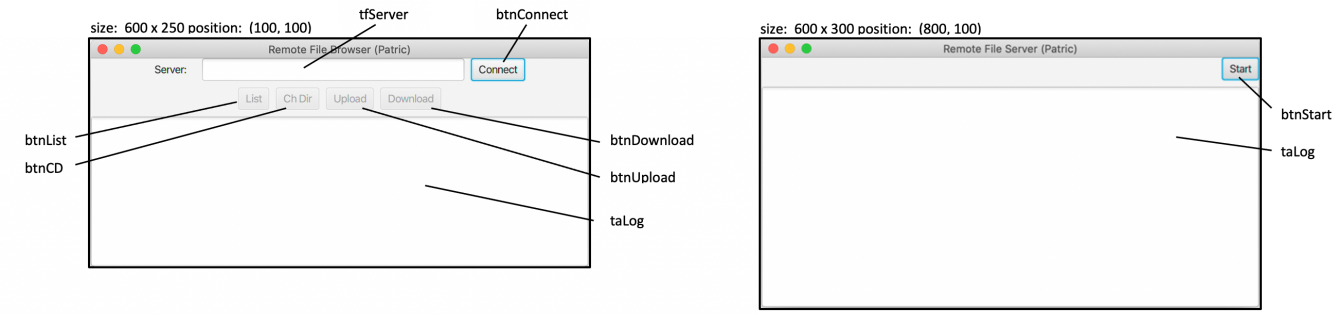
ISTE-121
Lab06/HW05 - Remote File Browsing

NOTE:

This is a combined Lab and HW. This means you will receive both Lab and HW credit for this assignment. When done, submit your work to **both** the Lab06 and HW05 Assignment folders. The lab portion will be assessed as usual (10 points) and the homework portion will be assessed against the gradesheet at the end of this document.

Objectives:

Write a Multi Threaded, Client / Server GUI that will provide remote file browsing on the server, from the client. It will also allow file upload and download of text and binary files.



NOTE: This is a (small) group project

You may attack this problem in a group. In fact, it is suggested that you do exactly that. Choose a partner, for a group of size 2 and get started. EACH of you must submit a solution to the Lab06 / HW05 Assignment folder.

NOTE: You have 2 weeks to complete this lab

Unlike most of the other labs in this course, this **does not** need to be done by the next lab period. In fact, many of you might not be able to do so. Instead, you have until Week 10 to complete this Lab/HW.

NOTE: You can run my solution

In today's downloads you will find RemoteFileBrowser.jar and RemoteFileServer.jar. These are my solutions to this problem. You can run them to get an idea of how things might work in your solution.

To run the RemoteFileBrowser:

In Windows, type:

```
runjava -jar RemoteFileBrowser.jar
```

On a Mac, type:

```
sh runjava -jar RemoteFileBrowser.jar
```

You can do similarly for the RemoteFileServer.

These require either file runjava.bat (PC) or runjava (Mac). As stated earlier in the course, these need to be either in your current directory, or better yet in your path. These are the files we discussed and wrote earlier. If you didn't retain it, instructions are available under Content → Support → How to Run with JavaFX Outside of jGRASP. Follow the instructions.

Part 1 - The GUIs

Above are my GUIs for the client (on the left) and the server (on the right). The client, called RemoteFileBrowser.java, has a place to type in the server's IP or name and a connect button. The connect button becomes a disconnect button once connected. The other buttons are disabled except when connected. It also has a TextArea as a place to log events.

The server, called RemoteFileServer.java, has a start button. The start button becomes a stop button when the server is started. If stopped, the button becomes a start button again. The server also has a log (TextArea).

NOTE: the title bar contains the author's name. Also, suggested sizes and positions are given. If you use these, then when you run both of these programs, the client appears to the left of the server, for convenience.

NOTE: Setting the Position of the Window

To do this, use stage.setX() and stage.setY() just before stage.show();

You do not have to use this GUI design. It is only an example. You **may** use this design and you **must** provide the same functionality. Get your GUIs designed and implement them to the point that they appear as you want and where you want on the screen.

GUI Hints

The Logs (taLog) in both the client and the server behave as follows:

- As words are painted in the log, if the line becomes too long to fit, the line is wrapped on a word boundary. See setWrapText() in the TextArea class in the JavaFX API.

Part 2 - Protocol Design

You must design a protocol for client/server communication. This means deciding what types of streams to use over the socket (you can only choose one). It also means deciding what messages are sent, and how they are formatted. Are they Strings, Integers, or what? Also, when a command is sent from the client to the server, what will the response be? What if there is an error? How will each party know how much data is being sent? If it is always a fixed size, great. What if the size can vary?

Part 3 - Directory Management

For the "List" and "Ch Dir" buttons, the server will have to keep track of the current directory for the client. This may be different for each client. I suggest you maintain a String variable (currentDir) which is the full, absolute pathname of the current directory. Then, when the user asks to change directories,

append the relative name of the new directory to currentDir after a file separator. See the File class for how to get the file separator in a system-independent manner.

Also, if the user does a “Ch Dir” to the directory ".." (one level up), we do not want the ".." to be in the currentDir String. To take care of this, append the ".." to the currentDir, after a file separator, then create a File object for this new directory and call the getCanonicalPath() method of the File class to get the new value of currentDir. This will resolve the ".." properly.

Part 4 - File Names

When an upload or a download is chosen, you will need to know the filename for the data on both the client and the server.

- The client can ask the user to type in the name of the remote file (the file on the server) using a TextInputDialog (see the API).
- On the client’s machine, the client should present a FileChooser to allow the user to browse and select a file on the client’s machine. There are two cases here:
    1. When doing an upload - FileChooser will allow the user to choose the file to **open** and send to the server.
    2. When doing a download - FileChooser will allow the user to choose the file to **save** the downloaded data to.

Look up FileChooser. Look at showOpenDialog() and showSaveDialog().

On the server, be sure to always append the given remote file name to the currentDir, with a file separator in between, to get the name of the File on the server.

Part 5 - Design, Devise, Discuss

Discuss / Design the functional parts of the client and server code, so that they follow your protocol, and the client and server work well together.

Part 6 - Time to code, if time allows.

Your GUI should largely work (see Part 1). Now to add the button functionality. I suggest you start with “List”, then go on to “Ch Dir”, then do “Upload”, and then finally “Download”. Get each of these buttons to work before going on to the next button. This will make your code much more manageable to write (and grow).

For example, once “List” is working, you can go on to “Ch Dir”. You can test whether “Ch Dir” works by calling “List” after the “Ch Dir” is done and seeing if the files listed are correct.

HINT: You may want to use the log() and alert() methods from Day20’s lecture notes in your server where threading is present.

Submit your *.java file(s) to both the Lab06 and HW05 Assignment folders when your code is working properly.

ISTE-121 HW05 Gradesheet

Remote File Browsing

Criteria	Possible points	Earned points
Interface		
RemoteFileBrowser.java		
Client GUI is implemented correctly	5	
One button is **both** Connect and Disconnect	7	
Buttons are enabled/disabled at appropriate times	4	
Upload prompts for remote file, FileChooser for local file	5	
Download prompts for remote file, FileChooser for local file	5	
FileChooser for upload is an open dialog	3	
FileChooser for download is a save dialog	3	
All key events are logged so you can watch the protocol work	8	
Program meets stated requirements	10	
Subtotal (client)	**50**	
RemoteFileServer.java		
Server GUI is implemented correctly	5	
One button is **both** Start and Stop	7	
When stopped, no new connections are possible	5	
When stopped, existing connections continue to work	10	
Ch Dir and List work properly	5	
All key events are logged so you can watch the protocol work	8	
Program meets stated requirements	10	
Subtotal (server)	**50**	
Total points earned:	**100**	
Deduction violations after above grading		
-Xlint messages. Need a clean compile -2		
Deduction for program not following naming conventions		
Deduction for proper coding style not being used: indentation, use of white space for readability		
Deduction for missing JavaDoc documentation: file & methods -5		
Server contains inadequate in-code documentation -3		
Total Grade:	**100**	

Additional Comments: