

FIG. 3: Requesting path

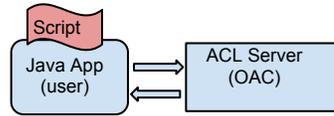


FIG. 4: Returning path to user so that the script may be sent to the OAC

clicks on the "Log" tab the display changes to a panel consisting of two buttons, a text field, and a large table. When the user clicks "Update" the table fills up with requests. The table consists of 1600 rows and five columns. Columns are as ordered: requester, time started, time completed, time cancelled, and request. When the user mouses over a row, the request will be displayed in the text field. The requester is a location within the lab that the request is made. The request is the actual ACL script requested. The user has the option to cancel the update by pressing the "cancel" button.

SOFTWARE/HARDWARE INTERFACE

The class files that make up my application are as follows: `AppGUI`, `Reading`, `Setting`, `ACL`, and `Convert`. The GUI class described above is self-evidently titled `AppGUI`. `AppGUI` handles displaying information and starting log and script tasks. `ACL`'s purpose is to consolidate all of the data being sent and received from `Reading` and `Setting` and then package the received data to be displayed in `AppGUI`.

Before the user can send down an ACL script to the OAC, a unique path must be designated. The user doesn't have to specify the path explicitly in the script; this process happens under the hood in the `ACL`, `Reading`, and `Setting` classes. This is a necessary step as multiple users can be accessing the same server. If two users try to send and receive data on the same path, only the first user will get readings. To designate a unique path, once the user clicks the "send script" button, an auto-

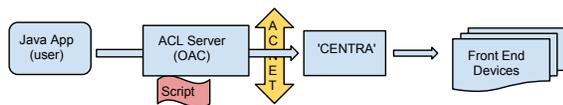


FIG. 5: Script is being sent down to the OAC where it will be converted into an ACNET message, which is then sent down to CENTRA. CENTRA will then perform the tasks.

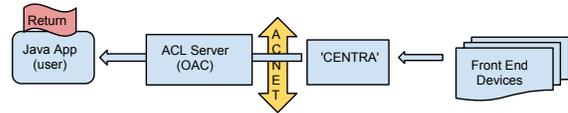


FIG. 6: Return data is sent back up through the same path.

matic process will begin. "Send script" in `AppGUI` invokes a runnable task that tells `ACL`, "Hey, I'm about to send a script down. Can I get a safe path?" as seen in Figure 3. Assuming nothing went wrong, the OAC will send back information to provide the user with a path as seen in Figure 4. Now that a safe path is established, the user can send the script down to the OAC. Next, `ACL` sends `ACL` information, including the script and defined variables, to the OAC. The OAC then converts the containing `ACL` information to an `ACNET` message and sends that down to the next node, "CENTRA" (Figure 5). `CENTRA` is a computer that runs an `ACLD` task, i.e it recognizes there is an `ACL` task and begins to run that task. That task was defined in the script the user wrote in the application; `CENTRA` then begins to fetch and/or send data from specified devices defined in the script. Once the task is completed, `CENTRA` packages that data in an `ACNET` message, then sends it up to the OAC. Next, the Java application will pull data from the OAC (Figure 6). On the software side, the Java application will read it in `Reading` and send it to `ACL`, which will then display that reading in `AppGUI`.

SCRIPTING

As stated in the previous section, this application comprises of multiple classes. `Reading` and `Setting` have a main function called "readMe" and "set," respectively. `ACL` makes one reading by calling "readMe" which returns an array of doubles; then the program enters a while-loop and takes another reading from "readMe." Again, the second reading returns an array of doubles. In addition to the `ACL` script, the two array of doubles from "readMe" are used as parameters for `Setting`'s "set" method in order to tell the OAC what task to execute. Here is some pseudo code¹ to give you an idea on how it looks:

```

//initial read
setparam1 = readMe(parameters)
while(no interruptions)
  //second read
  setparam2 = readMe(parameters)
  //sending script and instructions to OAC
  set(setparam1, setparam2, script)
  
```

¹ // denotes commented code, i.e., non-executable code

At this point, the OAC has instructions to carry out and will begin to do so; meanwhile within the application, a second "while loop" starts up and attempts to get a reading. Assuming the read was successful, `Reading` returns an array of doubles as previously stated. This array of doubles contains the information requested. To convert the array of doubles to a String, it must go through a conversion. To convert, ACL simply calls `Convert` which returns a String to be displayed in the GUI so the user views the data requested.

```
//start second loop
while(no interruptions)
    reply = readMe(parameters)
    replyString = convert(reply)
    if(replyString is good)
        display to screen
```

During the development the application is being hosted locally. Upon deployment, all of the class files and any other files that comprise the application will be moved to Fermilab's production drive where the application will be available globally.

LOGGING

As mentioned before, the application has a logging functionality to monitor ACL activity as seen in Figure 2. Filling up the table with information takes advantage of `Reading` much like sending a script task. In the OAC, a list of data is stored that contains logging information. Logging a table requires multiple calls to `Reading's readMe` method, which returns an array of doubles. One call to `readMe` will return a chunk of data correlating with the five column headers. The amount of data that can be returned by one call is limited by the OAC, so making multiple calls is necessary to fill table. The basic framework to fill the table looks something like:

```
while(no interruptions)
    readings = readMe(params)
    convert(readings)
    fillTable
```

The program will exit the while loop once the readings return nothing.

GOOGLE WEB TOOLKIT IMPLEMENTATION

The second stage of my project was to create an application for web browsers. Since the application above was written in Java, Linden proposed I create this application using Google Web Toolkit (GWT). GWT is an open source toolkit that lets developers write applications in Java but compile them as JavaScript allowing the application to be hosted locally or on a web server. As powerful as GWT is, it's



FIG. 7: Script being sent down to ACL server



FIG. 8: Returning values to client

surprisingly easy to use. To create a project with GWT, the user simply needs to type this command: `WebAppCreator -out projectName/path` while in `GoogleWebToolkit/Projects`. This will create a new project containing two essential folders: "client" and "server." These folders allow the project to be locally hosted and displayed on a webpage. To test the webpage, the user needs to enter another command in the specific project's folder: `ant devmode`. This launches GWT development mode where the user can launch a browser and see his/her's webpage. At this point, the Java code has been compiled as JavaScript. During the developmental stage, the application is being hosted locally. Upon deployment, the application will be hosted on Fermilab's Tomcat server allowing global access.

USING ACL VIEWER ON THE WEB

The GWT application process is similar to the Java application in that it communicates with another ACL server. This server was developed for URL access specifically; let's call this server URL. As seen in Figure 7, the client (web browser) sends data to the server folder (servlet) which directs that data to URL. Data from the application interfaces with URL by accessing a URL, `http://www-bd.fnal.gov/cgi-bin/acl.pl?acl=[ACL CODE]`. After the ACL task is completed, the data will be



FIG. 9: AcViewer Web GUI

processed and returned as seen in Figure 8.

The application developed in GWT has limited functionality as it can be accessed on any machine so long as the user has a Fermilab Service's account. That said, only some ACL commands are allowed, like `read`. Additionally, the GWT application does not have the logging functionality as seen in the Java application. Much like the Java application, the GWT application consists of two text areas and a button as seen in Figure 9. When the user clicks "send," script contained in the text area will be subjected to the process described in Figure 7 and Figure 8.

CONCLUSION

At first this project deemed daunting, as I had no experience with Java, GWT, or client-server communication. With the help of my mentor, Linden, the development process started to pick up speed. I quickly became comfortable with the Java language and learned a great deal on how users communicate with hardware throughout the lab. Both applications were developed to make ACL more accessible.

ACKNOWLEDGEMENTS

This work was supported in part by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists (WDTS) under the Community College Internships (CCI) Program. I'd like to thank Fermilab, the Accelerator Division, Linden Carmichael, Brian Hendricks, Arden Warner, Glen Johnson, and John DeVoy.

REFERENCES

-
- [1] J. Patrick, *Fermilab Control System ("ACNET")*. Batavia, Illinois, Feb 17, 2005.
 - [2] Andrey Petrov, Fermi National Accelerator Laboratory, Accelerator Controls Department. *Beyond ACNET: Evolution of Accelerator Control System at Fermilab*. SLAC, March 17, 2009.