VISA Communication Tool Project Progress Report

Chenliang Wang[1] and Wenbo Hou[1]

[1]Oregon State University

May 12, 2017

Abstract

This document describes the current state of the project, VISA Communication Tool. Each team members will state their contributions to this project. Since we lost one teammate, the development team rescoped the project and focused on the left work in last few weeks. This document will summarize efforts done by developers in last few weeks and provide a sort evaluation of the project.

Contents

1 Project Overview

The VISA Communication Tool is an instrument manage software designed for Test & Measurement Devices. It implements PyVISA and QCompiler to provide a quick and convenient instrument operation interface. The software supports device sensitive code auto-complete and documentation query service to help engineers to get rid of the annoying programming manual. Besides basic communication with the instrument, the software also allows user to import a pre-written script with standard instructions to improve work efficiency.

2 Project Progress

The development team divided the whole project into five parts based on different functionalities: Graphical User Interface, Sent Commands History, Code Auto-complete, Documentation Query Tool, and Communication API. So far, developers have only finished Code Auto-complete and Documentation Query Tool. If every thing went well, the development team would optimize the Graphical User Interface before EXPO. However, one of our teammates who is charging of Communication API was kicked off due to the limited contribution. The development team had to focus on building Communication API and give up the optimization work.

2.1 GUI Design

2.1.1 Task Review

From the objectives of the project and development team discuss with clients, one of the tasks for this project is to create a friendly user interface instead of the current one that clunky interfaces. As there is no any specific requirement, development team decided to make a GUI design first.

2.1.2 Current Progress

In order to match the clients' requirement, development team created the GUI design in the Fall term. The GUI design is through with Balsamiq Mockups which is really a good tool for GUI design. Development team create the GUI with a IP information area; a navigation bar, a big area to display the input and output; a command area for entering the command, functional button for writing and query beside the command enter area; command choice area under the command enter area, it will display when use enter the key word of the command; and the feedback information is at the bottom. After development team have done the GUI design, they sent the design to clients and get some feedback. After development team changed, clients agree with the design. The feedback from clients will be shown on the next part of interface design of first user study.

2.1.3 Interface Design First User Study

This part is only for interface design of first user study. After development team send the GUI design to clients, they gave some feedback. As beginning design, development team design only one command send button with both write and query. Users can use that switch the mode between writing and query. Also, there will be a "W" or "Q" in front of the command line to separate the mode is write or query. However, the clients think it is not friendly, they gave the advance to put two button one is to write, and another for queries. Therefore, development team delete the switch and change to two buttons. All, in all, based on the GUI design, they agree with this design that a good first user study. The GUI implement is based on this design.

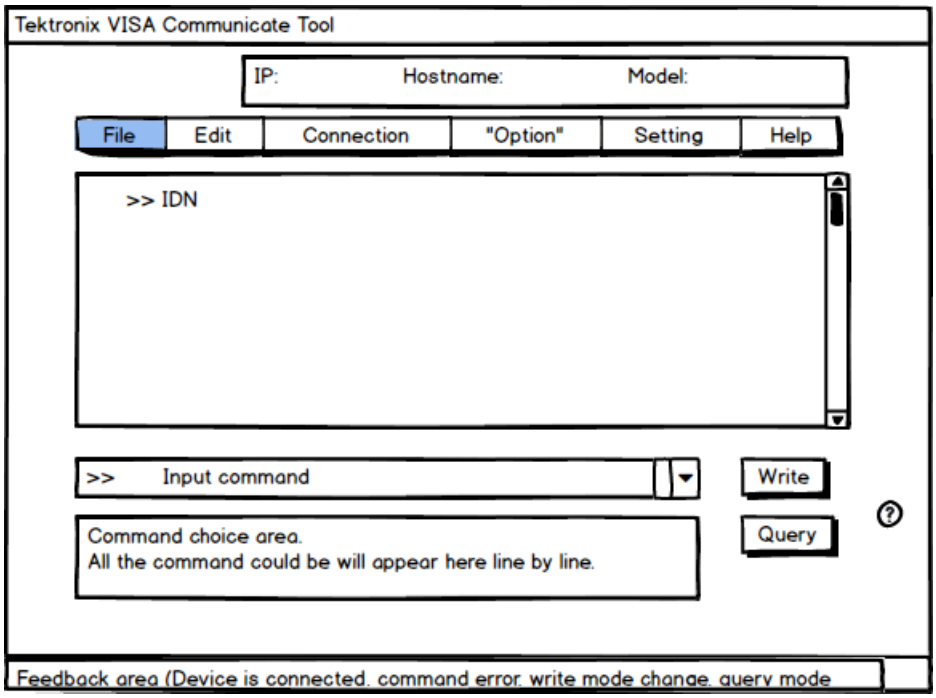


Figure 1: The image of GUI design

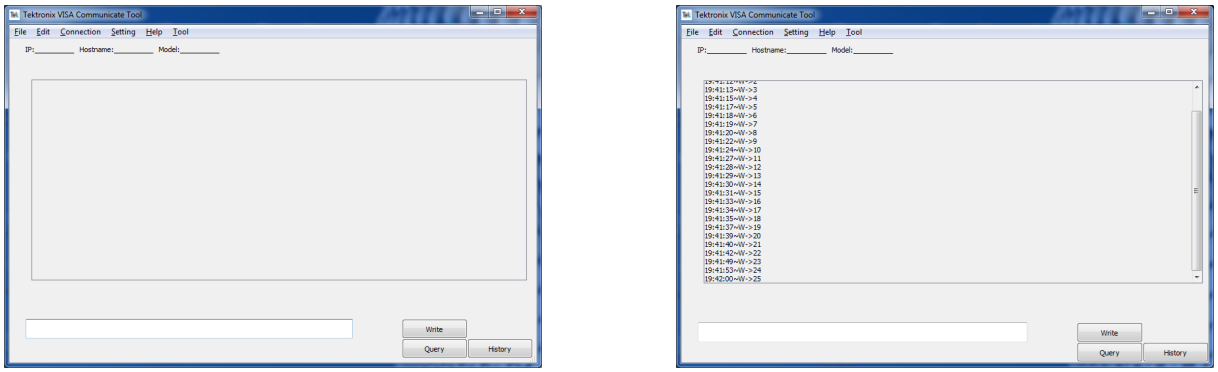
2.2 GUI Implement

2.2.1 Task Review

After development team finish the GUI design, they need to implement the GUI by coding. One requirement for the GUI implement is based on Python. As development team finish the GUI design before and clients agree with their design, the implement will be based on the design.

2.2.2 Current Progress

For the GUI implement, development team decide the coding environment that use PyCharm Python3 and PyQt5. Through the whole process, development team create the general GUI first and add the more items and functions one by one. Until now, they have a main interface and 3 sub windows. For the main interface, development team have the navigation bar on the top with some options such as exit, IP connection, command query, etc. The big input and output area can display the command input different by writing and query with W or Q in front. Also, there is time display before the writing or query signal, it can make users know when they send the command. There are three buttons with “Write”, “Query”, and “History”. The “Write” and “Query” buttons send the command; the “History” button can open a sub window which displays the sent 25 commands. For the sub windows, there are 3 sub windows now. History sub window, query sub window, and IP connection sub window. The details of each sub window will be in their self-part later.



(a) Initial

(b) After some sample Input

Figure 2: Software Main Interface

2.3 History

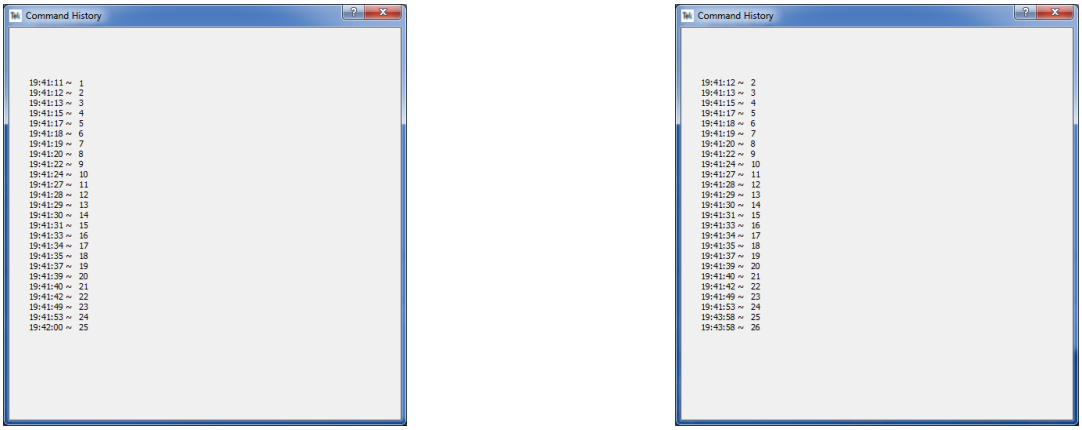
2.3.1 Task Review

25 recently sent commands history is the one part what development team need to do. As the clients' requirement, the software can display the recently 25 sent commands. When users want to check the commands what they sent before, it should be somewhere can see the command history.

2.3.2 Current Progress

The command history is displayed in a sub window. The history sub window is to display the recently 25 commands. On the main interface, there is a button called "History". When users click the button, or use shortcut (Ctrl + h), the history sub window will be opened and users can see the recently sent commands. The commands what the users sent will display from top to down. As the 25 recently commands, when users send the 26th command, the first command will be removed from the window. Also, the command history can see the time of when the command sent, it is good for users to find the command.

To test this part, developers sent 25 commands and check all of them can be all display on the history sub window. Then developers send more commands and the command history still keep 25 recently commands based on one command in, the old one remove.



(a) With 25 Commands

(b) After add 26th Command

Figure 3: Recently Sent Commands History

2.4 IP Connection

2.4.1 Task Review

IP Connection is the function what users need to use connect the PC and devices. It could be connected by two ways. One way is input the target IP address and port number for connection; another way is scans the local network and choice one target device for connection.

2.4.2 Current Progress

For this part, development team create the sub window for IP connection. In this window, it includes the IP address and a port number input area; and the local network scan area. The "IP Connection" button is for users with connection through IP address and port number. The "IP Scan" button is for users to scan the local network. As there are no devices what we can find through the network, developers can't test this function. Until now, the software has the IP interface and can display the local IP address, hostname and few network IP address. As this situation, developers use the software through the cable to connect the software and devices. In the future, if development team can find the devices, they will test and debug for this part.

Figure 4: The IP sub window

2.5 Code Auto-complete

2.5.1 Task Review

From the project proposal and discussions with clients, one of concerns of developing the new VISA communication software is to free users from the programmer manual. Our clients, Tektronix engineers stated that VISA command syntax is too complex to memorize. It was not efficient to referring the coding documentation when users operate instruments through the old software. Inspired by main-streaming IDE, our clients want a new software with the command auto-complete feature.

2.5.2 Current Progress

The development team has built two versions of code auto-complete script. The first one was a drop-down list with selectable strings below the input line. Unfortunately, it was not compatible with Graphical User Interface so that the development team abandoned that. The second code auto-complete script was built with **QCompleter**,a standard in-line completion toolkit from PyQt5. **QCompleter** handle the auto-complete pipeline in an excellent way, and developers only need to provide its reference, the syntax library. Since there are two command groups involved in

this project, the development team assigned two different icons to them. When the code auto-complete is trigged, an icon will show in front of the command.

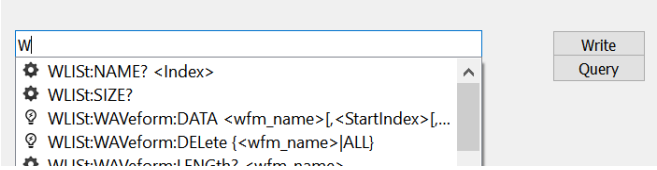


Figure 5: Icon associated commands in QCompleter

When testing the code auto-complete script, the development team noticed that it took quite long time to initialize the software. The development team found that the software spend much time on reading the syntax library. Thus, the development team made two improvements. First, developers implemented a new CSV reader module, pandas. This module is built for reading massive data, which makes data reading more quickly. Second, the development team chose to initial **QCompleter**'s reference dynamically. In fact, there are different types of Test&Measurement instruments that has different commands. It is time-consuming to read all command syntax together. Consequently, the development team modified the syntax reading function to read command syntax from source files based on the connected instrument type. If no instrument was detected, the software would load command syntax for Arbitrary Wave Generator (AWG) as default. By doing these two improvements, the software's initialization time became shorter than before.

```
......
#Device Identification Call
def idn_connected_device(self):
try:
    temp_com = CommunicationInterface()
    self.connected_device = temp_com.device_identification()
except:
    self.connected_device = "AWG1224"

# Active code auto-complete based on instrument type
def active_extention(self):
    self.extention = CodeAC(self.textInput, self.connected_device)
    self.extention.active_script()
    ......
```

Listing 1: Code Auto Complete Initialization with Device Identification

2.6 Document Query

2.6.1 Task Review

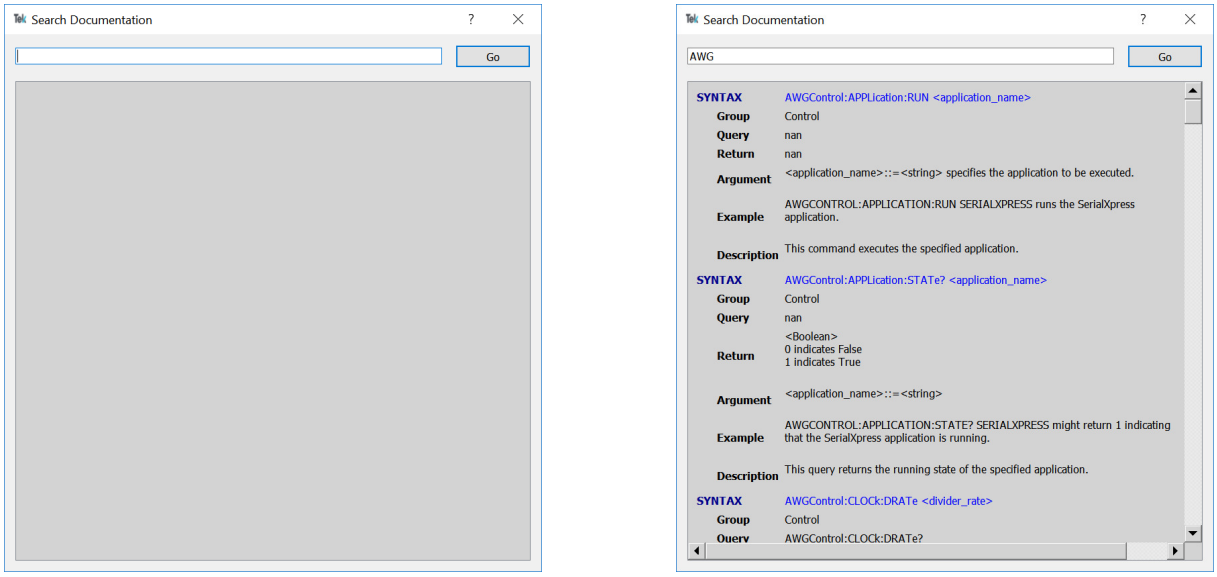
One of the software design requirements is building a document querying tool. The document querying tool allows users to retrieve information about a command or commands containing typed in keywords.

2.6.2 Current Progress

Based on the design document, the querying tool exists in a separate window that linked with a menu-bar option. The development team did not actually create a window for this querying tool because of limited knowledge about PyQt. Instead, developers created a **dialog** to hold the querying tool. There are two parts in this dialog: the searching area and the display area. Considering that multiple commands will be queried, the development team used the grid layout in the dialog. The first row hold the text input and the search button. The following rows

are reserved for query results. Similar to the code auto complete, the querying tool chose to read specific command library based on the type of connected instrument to optimize the memory usage and processing speed. The module, **pandas** involves in this part, either.

The biggest problem in building the querying tool was how to display the queried results explicitly. Since the searching algorithm only returned a list of strings, directly placing strings in the grid led a chaotic view. Thus, developers had to applied a style sheet on the dialog. Inspired by the formatted programmer manual from the clients, developers built an information block for the quired result and placed them into the grid. See figures below:



(a) Initial Dialog

(b) After search Dialog

Figure 6: Overview of Document Querying Tool

2.7 Communicate with Instrument

Task Review

As an instrument operation software, the communication interface is necessary. The original requirements of the communication interface contains more than sending instructions to the connected instrument. Due to the fact that the communication interface developer left the team, other developers had to responsibility for this part. Due to the limited knowledge and time, the remaining developers evaluated objectives in the communication interface and decided to accomplish objectives based on their priorities.

2.7.1 Current Progress

First of all, the development team had to complete the fundamental requirements, sending instructions and collecting return messages. Although developers spent weeks on researching PyVISA, they encountered with an obstacle, the configuration of NI-VISA library. In early development period, developers did not notice that PyVISA was an implementation of NI-VISA library. Consequently, developers failed to communicate with the instrument with only PyVISA package. After install NI-VISA library, the development team followed the instructions from PyVISA and tested the simple device identification call with Tektronix DPO4034 Mixed Signal Oscilloscope. Fortunately, developers got the correct instrument serial number from the oscilloscope.

```
# Get the serial number of the connected instrument
def device_identification(self):
    return self.my_instrument.query('*IDN?')
```

Listing 2: The Device Identification Call

Then, the development team started to build communication APIs for the input line in the user interface. Based on PyVISA document, the development team built two basic kinds of communication API: *write* and *query*. *write* allows users to send instruction to the instrument, and *query* allows users to collect the instrument state information or test results from the instruments.

2.7.2 Future Work

After achieving the fundamental interaction goals, the development team will work on extend functions of the communication interface. Currently, developers work on the work log and the command history because these two parts will uses several same functions. Then, the developers will code for the error handling and the debugger, which requires developers to research on PyVISA error handling policies.

2.8 Conclusion

Developers have been working on VISA Communication Tool since last fall term. During this time, developers learned lots of new and practical programming skills and acknowledge the importance of communication skills. So far, all MUST objectives of the software have been accomplished by developers. Besides that, some recommended features are also in progress like, the command history.

Recalling previous work on the software, developers considered that they did not work as expected. For example, the development process did not follow the proposed Schedule. Developers discussed the fact and generated a self-evaluation in last weekend. Developers considered that final score was 7 out of 10. There were three main reasons. First, the lack of communication with clients led the development to wrong directions for several times. Second, some developers were unable to finish assigned task on time, which slowed the development process. Third, developers did not spend enough time on this project. In future, developers will get rid of these disadvantages and work seriously.