Programming Public Safety: A Script to Help Save Lives

CS4991 Capstone Report, 2024

Hunter Brown
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
ghb6mt@virginia.edu

ABSTRACT

During my time at L3Harris, I worked on the back end of the radio network system where I created a configuration tool to assist in programming newly constructed radio network systems for law enforcement, firefighters, and EMS. Another team member and I created a Python script that read in configuration files and programmatically configured the towers in accordance with the specification sheet provided to the engineer setting up the tower. We were able to complete the project during the summer, turning a 2–3-week job into a 2–3-day job, in which our script was used to program all the new radio towers installed across the state of Florida. During this time, I also developed many professional skills enabling me to learn to document software, give demos in front of dozens of people, and work in a team on a strict time schedule for a release. While this project was successful at a basic level, further development is needed in the user interface, configuration settings, and the script versatility, allowing it to update as well as configure new towers.

1. INTRODUCTION

When I started at L3Harris, I was introduced to many of the different technologies they produce to best equip emergency services to do their job safely and effectively. They had begun work on the State of Florida Law Enforcement Radio System (SLERS) but times for configuration of new towers were very slow, taking an engineer 2-3 weeks to set up the tower completely. This was done as manual work, so an engineer could only do one tower at a time, along with the potential for mistakes which would take more time.

The brains of the radio towers being built in the SLERS region were called Enterprise Network Managers (ENM). The ENM controlled all the logic for trunking and directing digital signals to the correct handhelds and car radio systems. Being the core component of a large radio tower system, that is a part of a larger radio network system, the configuration of the network tower was done exactly to specification for responsiveness and interconnectivity with the rest of the network system.

When I arrived, there was a need to increase the productivity of programming the ENM's for this region, as a faster delivery of the product would allow the team to work on new projects along with saving the company time and money spent configuring ENM's for the region. Originally, several engineers had to be taken from a team to manually configure these ENM's in the factory by hand, taking away their work on other projects and impacting the productivity of a team. This created the necessity of creating a program that would allow the ENM's to be programmatically configured.

2. RELATED WORKS

A key technology we utilized was the creation of a RESTful API inside the ENM. RedHat describes a RESTful API as a programming interface that adheres to the REST architectural style of representational state transfer [1]. The RESTful API utilized HTTP requests, specifically POSTs and PUTs to send data to the ENM over the internet and update its configuration settings. The creation of the API by the ENM's manufacturer gave us the ability to programmatically interact with the ENM with HTTP requests. With it being a 1.0 version, we were able to directly interact with the manufacturer, Cisco, and ask for features to be implemented for our work and report bugs that we had encountered.

Another key technology we utilized to complete the project was use of the Pandas data frames inside the Python programming language. PyPI describes Pandas as a data analysis toolkit built for Python, but we were able to use its data frame features, which were very similar to an Excel sheet [2]. We used it to get the data in our program into organized objects that we could control for each portion needed for the API calls.

3. PROJECT DESIGN

When designing our project, we needed to identify what our client, being ourselves, needed along with clearly defining clear specifications to follow along the way. During this process we encountered some challenges and adapted solutions to overcome them.

3.1 Requirements

3.1.1 Client Needs

Our client for this script was ourselves. The script was for in-house use by the engineers who setup the radio towers to replace their manual configuration with our script for automated configuration. Our engineers at the time were spending 2+ weeks per ENM, manually entering data into its interface. This script needed to be compiled into an executable that could be run on any Windows system on weak hardware for both our engineers in the factory and for technicians in the field to be able to use it.

Along with the script, there was a need for internal documentation for the engineers/ field technicians to know how to use the script. They also needed internal documentation of the development of the script and how it works at a program level.

3.2 Key Components

3.2.1 Specifications

The specifications for our script were loosely defined other than the type of documents that it should be able to read in and the different configurable options. The document to be read in was an Excel sheet which contained roughly 10 pages in which we needed to use 5. We needed to scrape key information from these sheets including device names, device functionality, and device connections.

We were required to develop an interface for the user to interact with the script, but there was no specification whether that be done through the command line or a GUI interface.

The documentation specification included an internal company document for instructions for our engineers and field technicians to use the script. Also, we were required to document aspects of our development and how it works internally, along with robust testing documents for the testing part of the script.

3.2.2 Challenges

We faced two key challenges in our development: cleaning data and a buggy API for the ENM. The sheet that contained all the data was developed many years prior for a different system of radio towers, not for the current system. This left lots of legacy information and formatting that the engineers could read and correct on their own. The

legacy conventions also did not include the new naming conventions for devices on the ENM. Adding these devices was the core of our script.

We also were challenged by the RESTful API that was created for the ENM. This API was brand new and was being developed by Cisco directly for this project. The documentation for the API was not done well, which inhibited our ability to make progress efficiently. Some of the functions did not work as documented, which caused issues in development that we had to meet with Cisco to fix. We also were missing functionality in the API that would have been useful and made our script more robust and automated even more of the configuration process.

3.2.3 Solutions

For solving the issues with the data sheet, we had to read in extra information to apply the new naming conventions along with standard string cleaning. This added lots of extra hard coded maps in our script and caused us to add more to our configuration file, which became difficult to use unless one knew exactly how the script was going to handle it.

To solve our API problem, we had to go through many hoops. For many of the functions, the lack of documentation along with incorrect behavior according to the documentation required us to go through each of the functions and test functionality and understand exactly how the input was expected and what effects it would have on the ENM. To learn the input pattern and functionality, we had to reverse engineer useful functions that were hidden and undocumented.

4. RESULTS

We were able to programmatically read in the Excel file and direct the input into the Pandas data frames. With the forms being formatted similarly to the Excel sheet, they fit well into the frames, and it was easy to visualize exactly how they would translate from the sheet into our program. We were able to use the tools in the Pandas framework to clean our data, which included removing whitespace and cleaning up and removing null values in our data. Formatting in the data frames allowed us to use the ENM's API calls directly with the frames, calling a simple "toMap" function which fed the data in a JSON format packaged in a HTTP request that the API expected.

For our UI, we first wanted to create a graphical user interface, but found that a command line interface was all that was necessary since there was not much interaction other than starting the script and updating the configuration file with the intended settings. This was done with a clickable executable that launched a command line window and started the script which would open the configuration file, allow the user to edit it, then save and run the script.

Our script was completed on time for an end of July launch to help with the creation of the SLERS radio network. This script cut down configuration times roughly 70% for our engineers and allowed multiple ENM's to be configured at once rather than one by one. We created robust documentation for both what the script does and the API that we reverse engineered.

5. CONCLUSION

We were able to complete our script for L3Harris by the end of August, in time for the deployment of more of the SLERS system. The script was able to be utilized by engineers in the manufacturing plant to increase productivity in configuring the ENM's. We created a command line driven UI for users to interact with and usability features such as automatic window creation for popups of configuration files and loading bars.

The completion of the script was immediately seen as an advantage as multiple ENM's could now be configured simultaneously by a single engineer, increasing throughput of ENM's. This allowed them to configure multiple ENM's at the same time, and at a much faster speed.

6. FUTURE WORK

We were not able to complete all the work that we wanted to do on the script during my time there, and the project was put on hold when I left. There was more functionality that the API had that we were unable to program for in my time there, which would have continued to reduce the amount of time engineers spent working on the ENM's. We had more we wanted to add to the API by requesting features from CISCO but were unable to have those meetings before the project was paused. There was also a desire for a better UI, specifically a graphical UI which would be more user friendly and more professional versus the command line format we currently have.

REFERENCES

[1] "What is a rest api?," Red Hat - We make open source technologies for the enterprise, https://www.redhat.com/en/topics/api/what-is-a-rest-api (accessed Feb. 21, 2024).

[2] "Pandas," PyPI, https://pypi.org/project/pandas/ (accessed Feb. 21, 2024).