DEVELOPING SONAR: AN INTERNAL CONFIGURATION AND DIAGNOSTIC APPLICATION

Douglas Mulka

Kettering University

Department of Electrical Engineering

LightGuide Inc.

Dr. Xuan Zhou

2021

AUTHOR NOTE

This senior thesis has been submitted as partial fulfillment of the graduation requirements of Kettering University needed to obtain a Bachelor of Science in (Student Degree Program). The conclusions and opinions expressed in this thesis are from myself and do not necessarily represent the position of Kettering University or anyone else affiliated with this culminating undergraduate experience.

Although this thesis represents the compilation of my own efforts, I would like to acknowledge and extend my sincere gratitude to the following individuals for their valuable time and assistance, without whom the completion of this thesis would not have been possible:

1. **Mikayla Ray**, Software Solutions Manager at LightGuide Inc. – For providing countless hours of support and mentoring throughout the project.
2. **William Sommerville**, CTO at LightGuide Inc. – For teaching me everything I know about AR manufacturing and supporting the creative direction of the project.
3. **Joe Fischette**, Software Solutions Engineer at LightGuide Inc. – For providing valuable assets and advice throughout the development process.
4. **Xuan Zhou**, Associate professor of Electrical Engineering at Kettering University and my thesis advisor.
5. The whole **LightGuide Team** – For giving me learning opportunities throughout my co-op experience and allowing me to pursue this thesis.
6. My parents **Jim and Janette Mulka** – For always believing in my potential and giving me the foundation to be successful.

ABSTRACT

Sonar is an application that holds all the tools necessary to configure and troubleshoot LightGuide Systems. The application can configure LightGuide PCs according to the company standards. Using Sonar to configure a LightGuide System will significantly decrease the time required to integrate a system. This time can be saved on every system sold by LightGuide. The application also provides the user with diagnostic information about the current system. This idea was explored and worked on previously by other company interns but was not evaluated or deployed. It was the goal of the author to revive this project and make it a useful tool for the company. This thesis outlines the process of reviving and developing Phase I of this application, as well as provides information about the next steps for the project.

Keywords: software development, configuration application, diagnostic application, WinForms, WPF, VB.NET, Visual Studio, Industry 4.0, augmented reality manufacturing, Command Prompt, PowerShell, Windows 10, Windows Settings, user interface, unmanaged API, platform invocation, WinForms Controls, delegate function, Windows Registry, .JSON, serialize, Class, Module, Method, Function, Sub Procedure

TABLE OF CONTENTS

I. INTRODUCTION

LightGuide Inc. is an Industry 4.0 software company that specializes in making augmented reality solutions to improve manufacturing processes. The main product that LightGuide sells is a software platform integrated on custom hardware, individual functional units are referred to as a LightGuide System. This system is comprised of a standard Windows PC, acting as the brain, and a combination of input devices and output devices. These input and output devices are off-the-shelf products like projectors, 3D/machine vision cameras, barcode scanners, PLCs, torque tools, and any other machine that passes information using a widely recognized industrial communication protocol. However, the scope of this thesis will focus on the Windows PC and the configuration necessary to transform it from a normal PC into a LightGuide System.

The Integration Team is responsible for configuring each PC sold by LightGuide. Currently, this configuration work is done manually. This process involves an Integrator performing all these actions by hand from a printed 5-page checklist and can be quite time consuming, especially when integrating multiple systems. The Integration Checklist in full is attached as Appendix A. This process is routine and lengthy, making it a perfect candidate for automation. There is a method currently used by the Integration Team to cut down on the time required to integrate a system, which is using a Windows backup image. The Integration Team has images of already configured PCs which they load onto a blank PC to configure of the items from the Integration Checklist. However, there are settings that the image cannot configure. Additionally, the image provides no verification that these items have been completed, requiring the Integrator to still manually go through the checklist to validate the PC has been configured to proper specifications. This

is where the Sonar application comes in, it seeks to solve the pains associated with manually configuring these PCs. By automating this process, the Integration Team can save valuable time preparing each system and potentially unlock the ability to ship out LightGuide Systems at an unprecedented rate.

Project History

The development of the application is not the first iteration of an application designed to assist with configuring and diagnosing LightGuide PCs. This project began in earnest prior to the authors employment with the company. This original version of Sonar, referred to as Legacy Sonar hereinafter, was developed by a previous employee but never caught on within the company. This was partially due to the application being incomplete and partially because it did not do anything that an Integrator couldn't do faster. The original application was also developed using the Windows Presentation Foundation (WPF) graphical interface, an interface that the company no longer uses and could no longer maintain.

At the time the author was hired into LightGuide, Legacy Sonar was all but obsolete. However, the original value proposition of an application that could aid in configuration and diagnostics remained desirable. The author saw this as an excellent opportunity to revive the Sonar project and turn it into a useful tool for the company.

Goals of the Project

To prevent the new version of Sonar from suffering the same fate as Legacy Sonar, the author defined some goals for the final product. Those goals are as follows:

1. The application will decrease the time required to configure a PC in-house, prior to deployment of the system.

2. The application will provide system information to aid in troubleshooting LightGuide Systems.
3. To develop the application in the .NET framework, using WinForms as the graphical interface to ensure continued support of the source code.
4. To salvage as many features from Legacy Sonar as possible.
5. The application will generate a report similar in format to the currently used Integration Checklist to certify that the required configuration actions have been completed as expected.
6. The application will "warn" the user if any actions could not be completed and instruct them to complete the remaining actions manually.
7. The application will have an attractive and intuitive UI.
8. The development of this application will serve as a training period for the author to better learn the development process at LightGuide.

Overview

In the remainder of this thesis, the author will describe the successes and shortcomings of the project development, as well as outline the subsequent `s of development to take place when the author starts as a full-time employee of LightGuide. Following this analysis, the author will outline the entire development process that took place from July-September 2021, also known as phase I. Finally, the author will conclude with the results at the completion of the project and supporting documents relevant to the development of Sonar.

II. CONCLUSIONS AND RECOMMENDATIONS

The development of Sonar was considered to be successful within the scope of this thesis. Within the 3 months dedicated to the development of Sonar, the author was able to complete or partially complete all the original project goals. However, the project is not in a state where it can be used as a permanent fixture of the LightGuide process. This is due to the lack of rigorous testing and it not looking professional or finalized. An apt comparison would be to call the present iteration of Sonar a first draft of the final application.

At the onset of the development process, it became evident that the scope would need to be limited in an effort to have a semi-functional product at the completion of the work term. This is due, in part, to the author's lack of previous development experience and because this work term was the author's final work term, leaving just 3 months to complete the project.

Phases of Development

After consulting with various LightGuide employees, the author decided to break the development of Sonar into 4 manageable phases to guarantee it's completion. The development phases are as follows:

- **Phase I: First Draft –** To produce a functional proof of concept to verify that Sonar truly is a project that can provide value to the company. This phase can be thought of as the alpha release.
- **Phase II: Beautification and Testing** – The main goal of this phase is to give the application a face lift to prepare it for use by the end user. This phase is

characterized by overhauling the UI to make it intuitive, responsive, and professional. After that, it will be rigorously tested to ensure the application is reliable. This phase can be thought of as the beta release

- **Phase III: Initial Release –** After it has been properly validated, Sonar will go through an initial release and be distributed to the Integrators for use in their day-to-day operations. Data will be collected in this phase to find ways to improve Sonar for the future.
- **Phase IV: Continuous Improvement –** This phase is reserved for implementing feedback collected following the initial release and additional feature implementation. As additional use cases pop up, they can be added to the source code to relive pains not previously identified in the initial value proposition.

Conclusions

Considering that the extent of this thesis concerns phase I of the project, Sonar will be evaluated as a proof of concept. The author believed that the greatest value for Sonar would be as a configuration and diagnostic tool, so those were the areas that were focused on in the development process. The 3 main features that were developed to satisfy those criteria were: the Configuration Page, the Diagnostic Page, and System Report.

The Configuration Page is where Sonar saw the greatest success as it solves the biggest pain in the integration process. However, due to the time constraint the Configuration Page does not do a full run through of the Integration Checklist, rather it configures the remaining checklist items that cannot be automatically set by a Windows image. In future releases of Sonar, the application will be able to configure every item on the Integration Checklist and provide verification. Another area for improvement on the

Configuration Page is the formatting of the Integration Document. Currently, it provides a generated document detailing the configuration status of all the checklist items but does not resemble the Integration Checklist or provide instructions on how to complete unconfigured actions manually. These are 2 things that will be implemented in a future release of Sonar.

The Diagnostic page holds all the features that were salvaged from Legacy Sonar. This page is a great resource for gaining valuable statistics pertaining to the current system but does not provide much insight based on this information. This tool would become significantly more powerful if this Diagnostic Page could intelligently make recommendations based on the information that it is collecting. Additionally, the formatting of this page could be improved. It displays information just for the sake of displaying information and does not necessarily pertain to actual troubleshooting use cases. The format of this page will be improved when some data can be collected on what information is most useful when troubleshooting systems.

The final major feature implemented in phase I was the System Report. This feature provides a very in-depth breakdown of all information that can be collected about the hardware and software on a system. Nevertheless, it too suffers from the same drawbacks as the Diagnostic Page. The information is not very readable to humans as it is a .JSON file intended for machine-to-machine data transfer and does not serve any purpose other than displaying information. The biggest improvement for this feature would be to format it so that it resembles the present checklist and is more easily understood by humans.

Sonar does a good job of proving the value of a configuration and diagnostic tool but needs to be improved to actually realize that value. As previously stated, there is plenty of room for expansion in each of the currently implemented features as well as the development of additional features in the coming phases of the project.

Recommendations

The future recommendations for this thesis pertain to the supplementary phases of development. The aforementioned conclusions will be implemented as a task of phase II because they mostly pertain to the user experience and usability of the product. It is quite important to focus on these items before the initial release of Sonar because if the software is not intuitive and easy to use it will never catch on as a productivity tool. Another main function of phase II will be to overhaul the UI to ensure it is consistent with the companies branding guidelines. The final main task in phase II will be to thoroughly test the features to make sure they perform as expected and actually provide value.

Phase III will be the initial release of Sonar and will be characterized by collecting data on how the software is used as well as what concrete improvements it provides. This data can then be fed back into the development process to guarantee that Sonar will make the task of configuration and troubleshooting easier. The information collected in this phase will then shape the formatting of the pages, emphasizing the features found to be the most helpful and refactoring features that may not provide much value.

The final phase of this project, phase IV, will be to implement additional features that were not present in the initial release. This phase will be perpetually ongoing as new

use cases for Sonar are established. Some features that could be implemented in this phase are display troubleshooting options, the ability to configure EDIDs, and the ability to be incorporated into the LightGuide software.

III. METHODS

The first step in developing any application is to choose an appropriate development environment and supporting graphical interface. At LightGuide, the most commonly used development environment for creating custom code applications is Visual Studio (VS). Legacy Sonar was developed in Visual Studio using VB.NET as the language and WPF as the graphical interface, so it makes sense to choose Visual Studio and VB.NET for the new version of Sonar. However, the main difference between the legacy version and the current iteration of Sonar is the graphical interface. WPF has the capacity for creating more attractive interfaces but uses the Extensible Application Markup Language (XAML) making it harder to maintain than WinForms. WinForms, on the other hand, relies on inheriting prebuilt Windows Controls, making it less flexible but much easier to work with. With that in mind, the decision was made to use VB.NET as the programming language and WinForms as the graphical framework. This made it possible to salvage a good chunk of the source code and migrate the front end from WPF to WinForms.

Pre-Development Planning

To combat scope creep, the author started by creating a Gannt Chart to define the timeline of phase I (Figure 1). This Gantt Chart proved to be incredibly helpful for keeping on track throughout the development process and was followed faithfully. Each

of the goals outlined in the Gantt Chart will be explained in further detail in the subsequent sections. In addition to the Gantt Chart, a Trello Board was also used for task management.

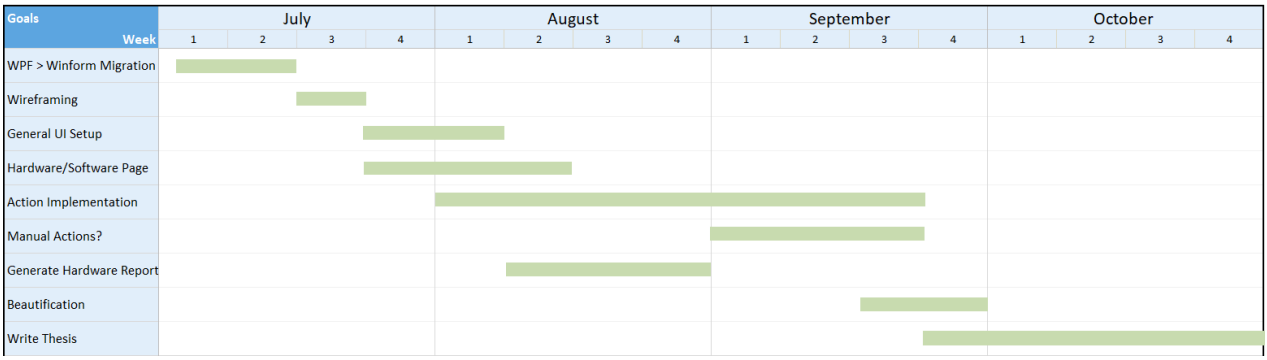

Figure 1: *Gannt Chart for phase I of Development put the Figure and tiles below the figure*

Wireframing & UI Design

Before programming could commence, a suitable user interface (UI) needed to be created. This was more in depth than just copying the Legacy Sonar UI because it was made it WPF; making it very difficult to replicate in WinForms. The best plan of action was to start from scratch and reimagine the UI for the new application. To reimagine the UI, a process called wire framing was implemented. This process involves creating a stripped-down framework of how the website should look. This makes it very easy to mockup, experiment, and modify designs before committing to one. Several possible designs were created and workshopped before finally settling on a design that would be able to accommodate the features in mind. For the Home Page, a Windows Control Panel

style was chosen so that it would be intuitive and user friendly since most people are comfortable using the Windows Control Panel (Figure 2).

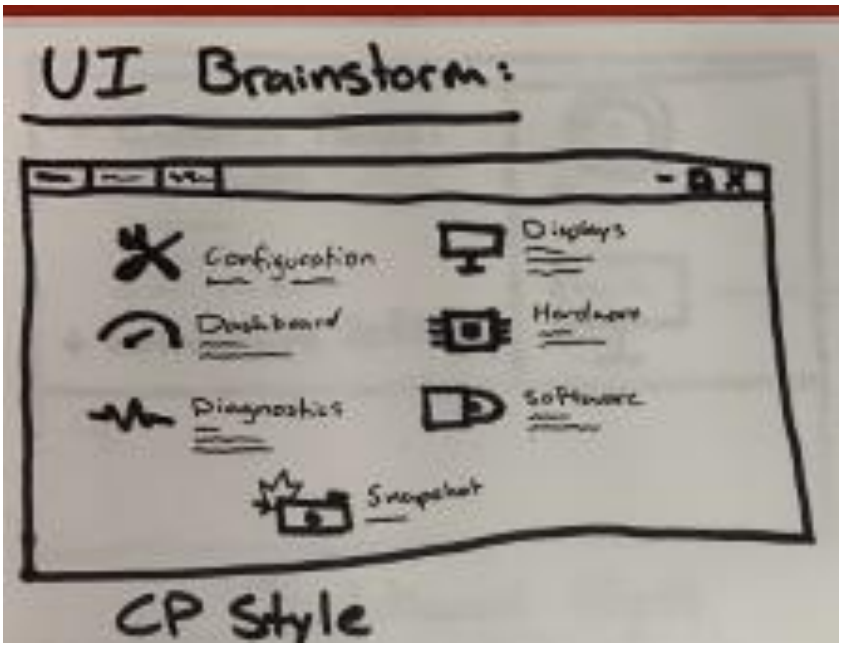

Figure 2: *Initial Hand-Drawn Design for the Control Panel Style Home Page*

After designing the Home Page, the Configuration and Diagnostic Pages needed to be created. These would have to be designed from scratch as there weren't any relevant Windows applications to borrow from. To make the wireframing process easier, the application Pencil was used to mock-up some interfaces. The advantage to using a software like Pencil is that it has several premade Windows Controls that can be dragged and dropped onto a canvas to quickly create designs. Since Sonar was using WinForms, it made sense to use a GUI prototyping tool that supported Windows Controls. After a few iterations, a preliminary design was created for both the Configuration and Diagnostics Page (Figure 3).

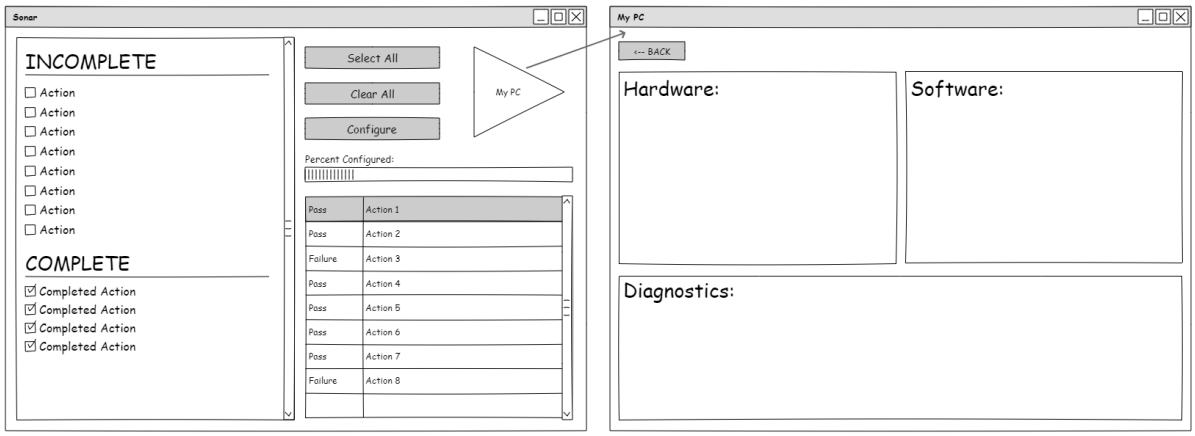

Figure 3: *Preliminary GUI Wireframe Made Using Pencil*

Visual Studio Project Initialization

After deciding on the framework and UI, the project itself needed to be created. LightGuide uses GitHub to allow for collaboration and version control, so this meant creating a repo for Sonar on GitHub. The repo was generated in Visual Studio 19, using VB.NET and WinForms. Once the Visual Studio Project was created there were still a few items to complete before code could be written.

The first of which is installing the requisite NuGet packages. Visual Studio uses NuGet Package Manager as a repository for controlling 3rd party add-ons which allows developers access to tons of features not native to VB.NET. Since a large part of this project requires modifying Windows Settings, it was necessary to install packages to aid in this task. One way to make these changes without using the Windows Interfaces is via Command Prompt (CMD) and PowerShell. To allow this VS Project to write to the Command Prompt and PowerShell the CommandLineParser and Microsoft.PowerShell.5.ReferenceAssemblies packages were installed. The next package was Newtonsoft.JSON. This package allows for classes to be serialized and deserialized

to .JSON files. This will be useful later when implementing a class for capturing the system information. Finally, all good software makes use of logging to aid in programming and debugging. The logger of choice at LightGuide is NLog. To get NLog into the project, the NLog and NLog.Windows.Forms packages were installed.

According to the LightGuide programming best practices, there are a few things that must be done at the onset of every programming venture. The first of which is establishing a Main Form. The Main Form is a Class that connects the UI to the code. The Main Form serves a couple purposes, to act as a container for the pages and to hold the menu strip. Firstly, the Main Form has a Panel Control which holds all the pages supported by Sonar. To achieve this, each Page is established as a UserControl and hot swapped into the Panel Control based on button clicks. When Sonar is launched the Home Page is shown in the Panel Control on the Main Form. When the Configuration Page or Diagnostic Page is clicked, the Home Page gets cleared and replaced with the respective page. The other main feature supported by the Main Form is to house the Menu Strip. The Menu Strip is a Windows Forms Control that allows the program to have a menu similar to any Windows application. The Menu Strip has an “Open” menu that allows Sonar to open related programs, such as: application logs, LightGuide Software, Cognex Software, Notepadd++, TeamViewer, Command Prompt and Powershell. The other option on the menu strip is “Help” which connects the user with documentation to help them use the software (Figure 4).

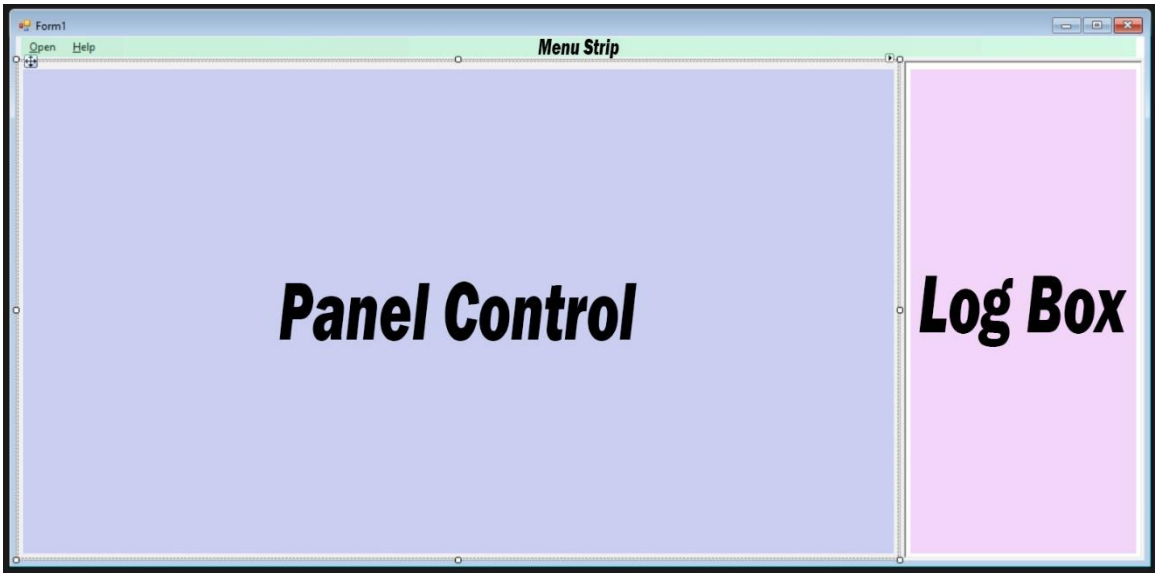

Figure 4: *Sections of the Main Form*

The next step is to establish a Main Module. A Module is a collection of code that is instantiated exactly once, is exclusively comprised of Shared members, and has the same lifetime as the entire program (Microsoft, 2021). The primary function of the Main Module is to hold any methods that need to be globally accessible. In Sonar, the Main Module holds all the functions for the configuration actions which will be discussed further in the Configuration Page Development section. The other primary function of the Main Module is to establish logging. LightGuide has a standardized Logging Class that is an extension of the methods provided by NLog. Sonar makes use of this Logging Class which is called by the Main Module.

WPF Migration

Since Legacy Sonar was developed in VB.NET and still contained relevant code it made sense to migrate as much as possible to the new code base. The main challenge associated with this migration is that Legacy Sonar used WPF, a very different graphical

interface. This means that any code that interacts with the UI will need to be completely changed. Fortunately, any code that was strictly business logic could be migrated with little to no refactoring.

This proved to be incredibly beneficial in creating the Diagnostic Page as most of those features were ripped from Legacy Sonar and re-skinned using WinForms Controls. The main value that Legacy Sonar provided was its ability to acquire information about the hardware and software on a given system. Not only could Legacy Sonar record and display this information, but it could also compare that file to the last time Legacy Sonar was ran and compare the differences. These features provide information that is extremely helpful when diagnosing problems with a system, so it was important to ensure that they were preserved in the new version on Sonar. The Classes that pertain to these features were easily migrated to the new VS Project. More information on the specifics of these classes will be provided in the Diagnostic Page Development section. The only changes required to make this code work were to remove all references to the WPF UI and replace them with equivalent controls in WinForms.

Home Page Development

The Home Page is the first thing that users see when opening the Sonar application, so it is important that it is clean and informative. For that reason, a Windows Control Panel style was adopted for the Home Page (Figure 5).

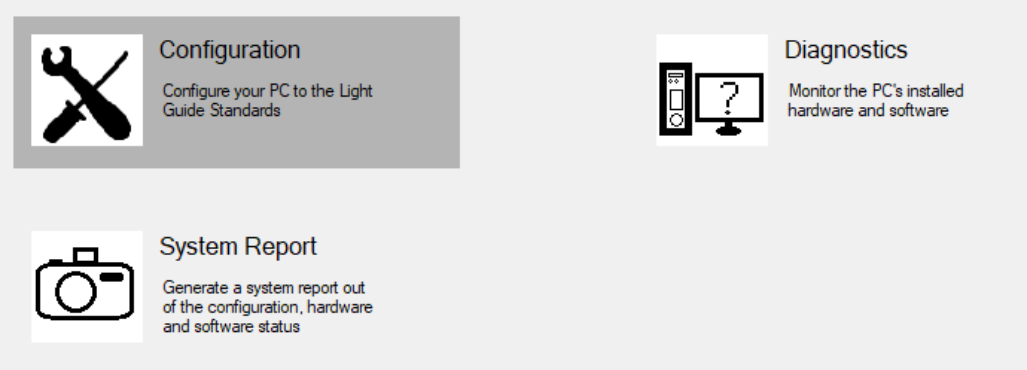

Figure 5: *Sonar Home Page Options*

Looking at this page the similarities to the Windows Control Panel become evident (Figure 6). The Control Panel displays each of its subsequent menus in a grid with a descriptive icon and a succinct description. This was used as the inspiration for the Sonar Home Page as virtually all users of Sonar will be incredibly comfortable with using the Control Panel. Since it displays items in a grid it will be easy to add additional options to the menu when more features are implemented.

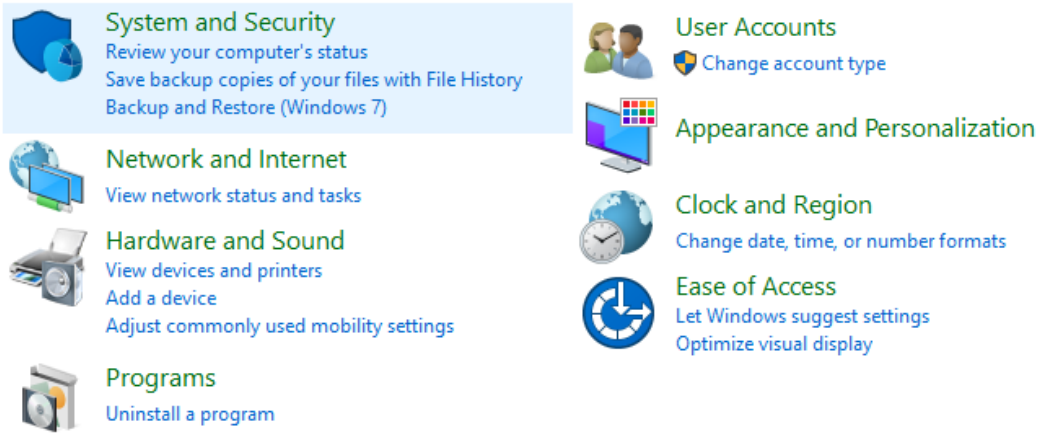

Figure 6: *Windows Control Panel Options*

The three Home Page options supported by this version of Sonar are: Configuration, Diagnostics, and System Report. The first two options will take you to their respective pages which will be explained in full in future sections. The third option, System Report, is what generates the .JSON file detailing information pertaining to the current system. When clicked, Sonar will show a SaveFileDialog Control prompting the user to choose a custom destination to save the System Report to. Behind the scenes, there is a serializable class called SystemInformation that saves a record of the current date, time, all software, and hardware on the system. This class is then serialized and saved to the chosen destination. The full system report is over 10,000 lines of system data, so an excerpt of the full report has been included as Appendix B.

Each of the "buttons" on the Home Page are not actually WinForms Button Controls, but are custom made controls (Figure 7). This was done because creating a custom control allows for significantly more customization than the default Button Controls. This custom control was developed by Joe Fischette, another LightGuide employee, for a separate project and was adapted to fit the needs of Sonar.

Figure 7: *ControlPanelIcon Custom Control*

This is the Custom Control, called “ControlPanelIcon”, that was used for the buttons on the Home Page. In its simplest form, it is a Picture Box and two Labels inside of a Panel. However, thanks to code behind the scenes it can appear and function as a button. When the code runs, it creates 3 of these Controls, adds in the appropriate graphic, and sets the text for the title and description Labels. This Control also has a couple Events that make it behave like a button. There is a Click Event that, when triggered, will execute the page switching code and a MouseOn Event that shades the button when it is hovered over. Custom Controls are used quite frequently in this project to give the application more functionality than is natively allowed in Visual Studio.

Diagnostic Page Development

When the user clicks on the Diagnostics Button on the Home Page they are taken to the Diagnostics Page. The main purpose of this page is display diagnostic information such as hardware and software configurations of the system. This page is essentially a graphical representation of the System Report described earlier. In fact, it makes use of the same serializable class to display the information. The SystemInformation class has 4 distinct parts: log time, installed programs, system software, and system hardware. The Diagnostics Page displays the system software and system hardware information in the tabs on the left. These two tabs, Hardware and Software, are made using a Tab Control to switch back and forth as well as more of those same custom button controls from earlier. This time the button control has been modified to act as a drop-down when clicked rather than a navigation button. This can be done simply by changing the ClickEvent. The drop-down feature allows the button to display additional information when clicked as shown for the Processor Button in Figure 8. This is the behavior for the buttons on the Hardware

Tab; they display information about the physical components of the system. The Software Tab does the same, instead it shows the software installed on the machine, the file path, the version number, size of the application, and when it was installed.

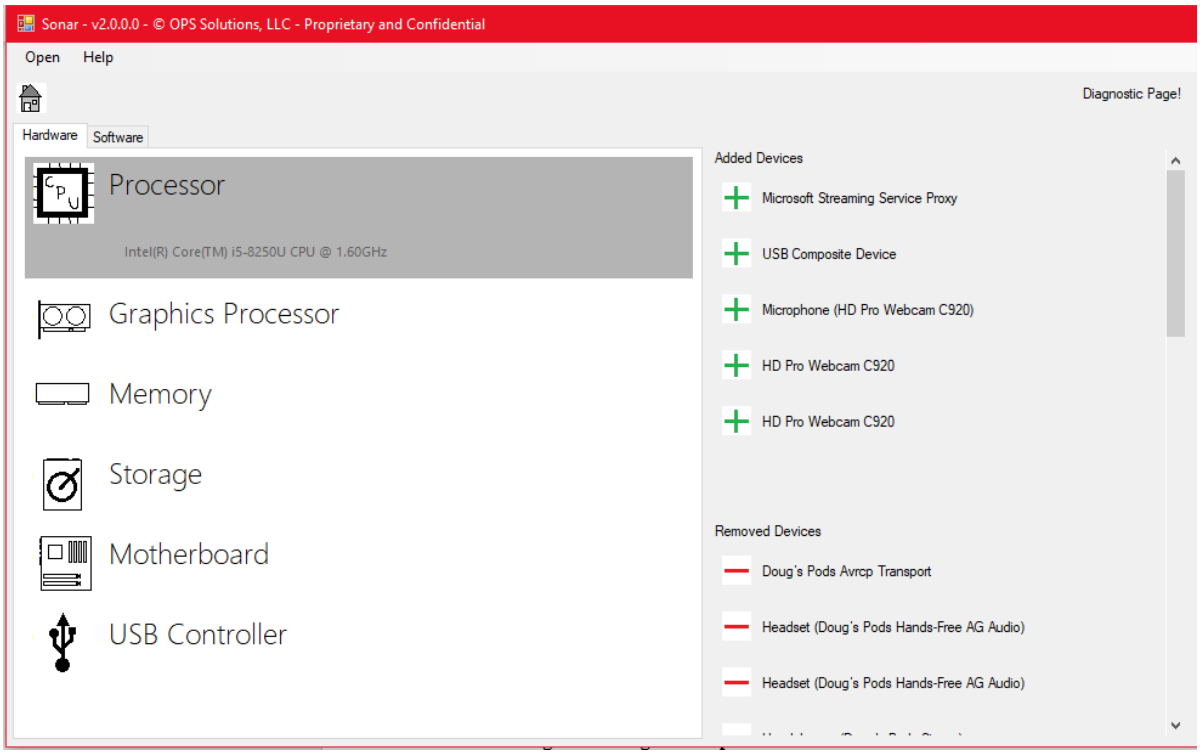

Figure 8: *Diagnostic Page with Hardware Information Displayed*

The other feature on this page is a pane that displays the added and removed devices from the system. Whenever Sonar is launched it saves a copy of the System Report and the next time Sonar is launched it compares these two instances of the System Report. Any differences between the System Reports are displayed here as either Added or Removed Devices. This is a beneficial Diagnostic Tool because when these LightGuide Systems are deployed there should be no changes made to their hardware or software. If a system spontaneously stops working it is always important to verify that

there have been no changes to the system made by the customer or customer contract service such as IT, this tool makes change tracking quite simple.

Most of these features have been repurposed from Legacy Sonar; these were the features that made up the bulk of the original program. Hardware information, software information, and device changes were all on separate pages in Legacy Sonar. Since they were similar features, and all used the same data it made sense to merge them into the same page (Figure 9).

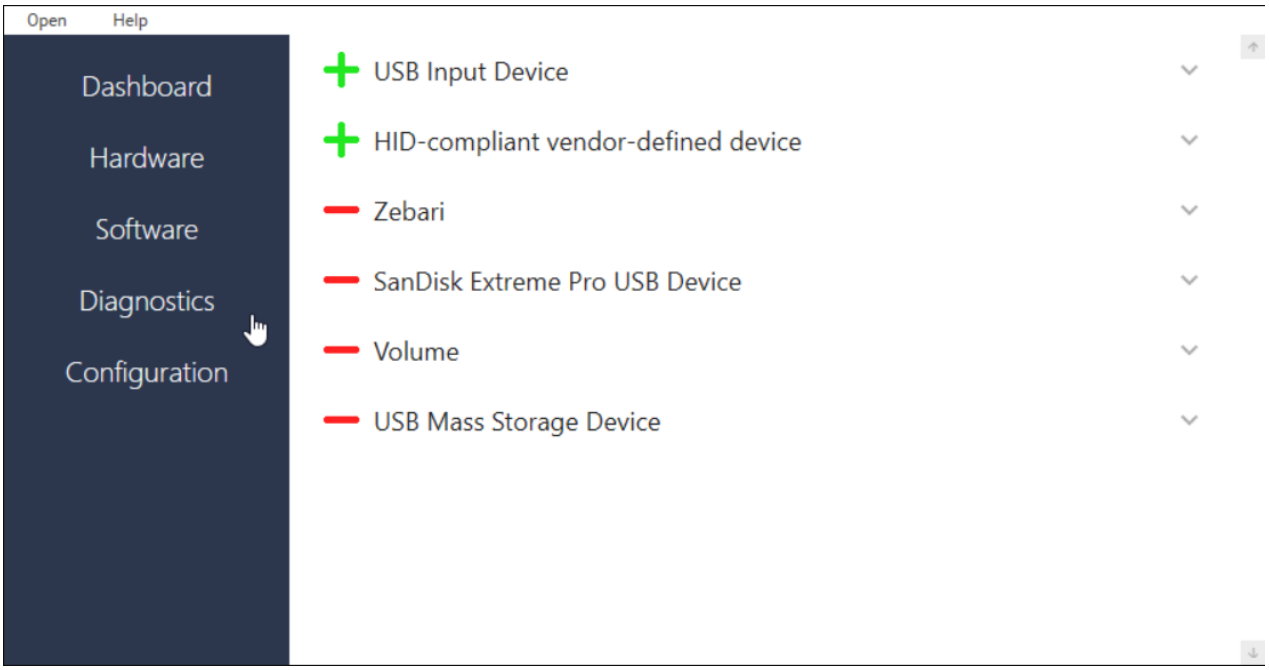

Figure 9: *Legacy Sonar Diagnostic Page*

Configuration Page Development

The final, and most involved page in this current iteration of Sonar is the Configuration Page. Clicking on the Configuration Button on the Home Page will take you to the Configuration Page (Figure 10).

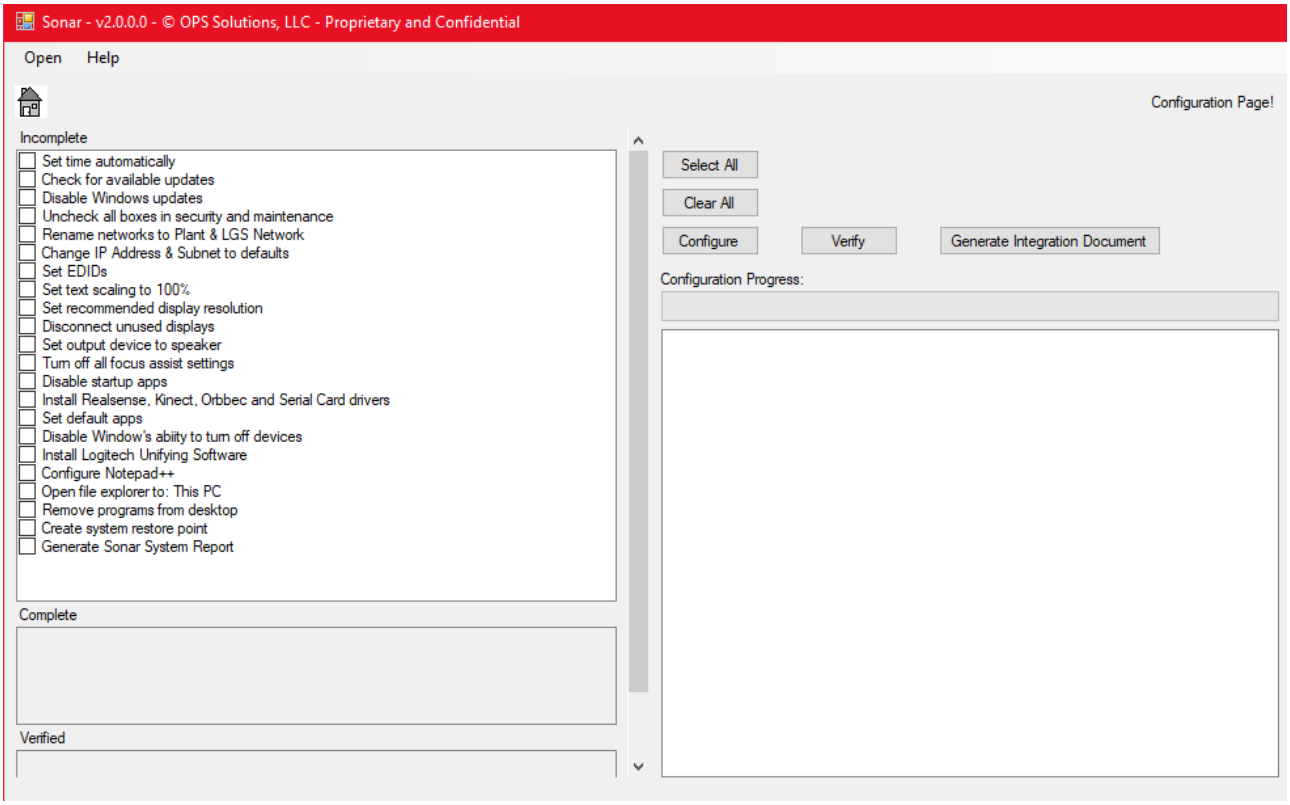

Figure 10: *Configuration Page of Sonar*

The purpose is to provide a clean and reliable interface for performing all the necessary configuration actions. You can select individual or multiple items from the Configuration Checklist to perform or verify. Each of the actions in the “Incomplete” section corresponds to an item on the Configuration Checklist. While there are only 22 items listed here, the full Configuration Checklist has over 100 actions that must be performed. This is because most PCs are not integrated from scratch, they are integrated from a pre-existing image that configures as much of the checklist as possible. Sonar assumes that the PC has been imaged already and seeks to configure the actions that remain after imaging a PC. In later phases of development, there will be a setting for PCs

that have been imaged and for PCs that haven't been imaged yet. These 22 actions were chosen because they are things that an image can't configure.

When the page first loads, all items start in the "Incomplete" section. From there a PreVerify() sub procedure is called to check if any of these actions have already been completed. If so, it moves them directly from the "Incomplete" section to the "Verified" section. From there, the user can select individual actions to perform by clicking on the checkboxes. The user also has the option to select all incomplete actions by clicking "Select All" or unselect all incomplete actions by clicking "Clear All." Once the desired actions have been selected, the user can perform them by clicking the "Configure" button. More information about how the program performs these actions will be given in the Action Implementation section. If the actions were completed successfully, then the action moves to the "Complete" section, if not then it stays in the "Incomplete" section. The actions in the "Complete" section have been performed but the program has not verified that the action has been performed correctly. At this point the user can press the "Verify" button to verify that each of the actions in the "Complete" section have been performed correctly. If the action can be successfully verified then it moves to the "Verified" section, if not then it moves back to the "Incomplete" section. After configuring the required actions, the user has the option to generate an Integration Document by clicking on the button of the same name. This document is essentially a certificate stating which actions have been verified and which are still incomplete. Figure 11 provides a detailed flow chart outlining the Configuration Page process.

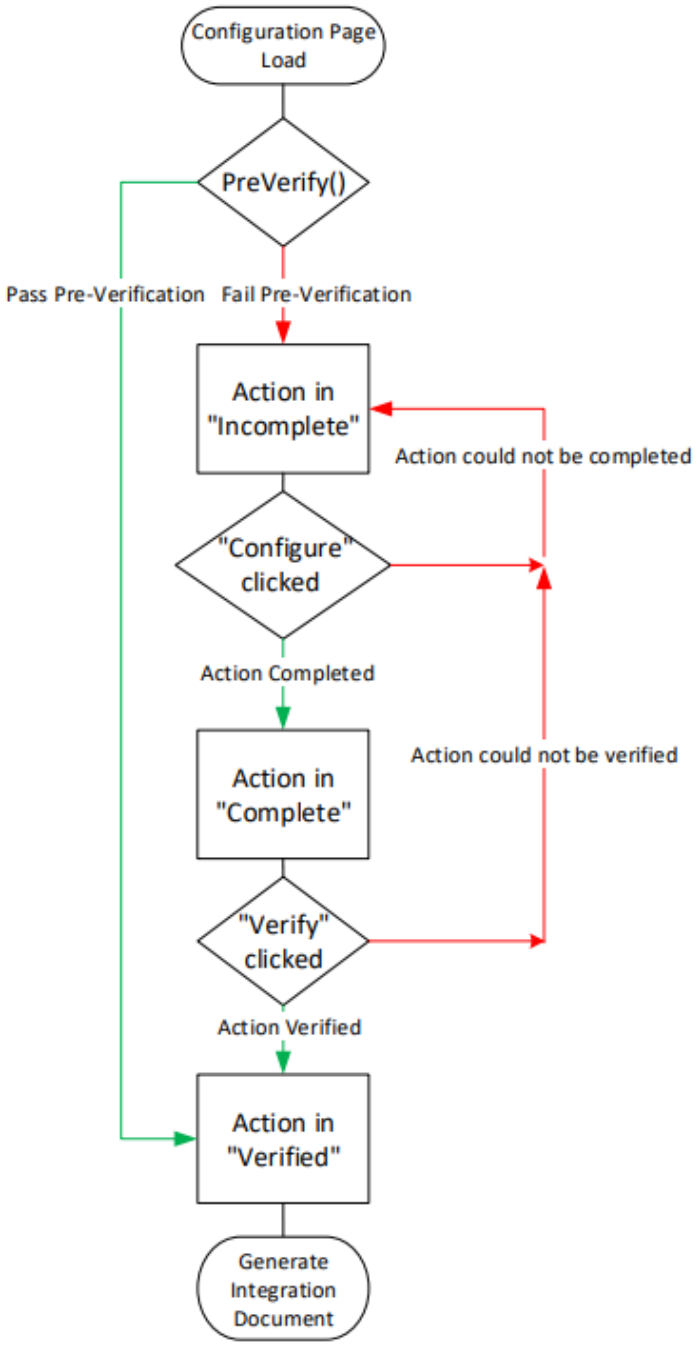

Figure 11: *Process Flow for the Configuration Page*

Past the main functionality of the page, there are a couple additional features on the Configuration Page. There is a progress bar that displays how many items have been

successfully verified. Additionally, there is an output window that passes useful information back to the user. Every time the user attempts to configure or verify an action, the result is displayed in the output window. The final feature of the Configuration Page is the Integration Document. When the button is clicked a text file is automatically generated and saved in the application logs. This file contains the name of the integrator, the date and time the actions were performed, a list of the actions that have been verified, and a list of the incomplete actions. An example of this document can be found in Appendix C.

The legacy version of Sonar also had a Configuration Page, but it was very obsolete when this project was picked back up. The Integration Checklist had gone through a couple of revisions since then, so the actions weren't accurate. Also, most of the actions were things now handled by the image. When development began on Sonar V2 a lot of focus was put on making this Configuration Page more beneficial. Looking at Figure 12, there are some similarities to the Legacy Sonar Configuration Page, but the overall layout has changed quite significantly. One of the changes made to make this happen was to consolidate the information. Legacy Sonar had multiple tabs, each containing actions, that separation made it difficult to configure all actions simultaneously. Legacy Sonar also did not have the ability to verify the completion of these actions or to generate a document certifying the completion.

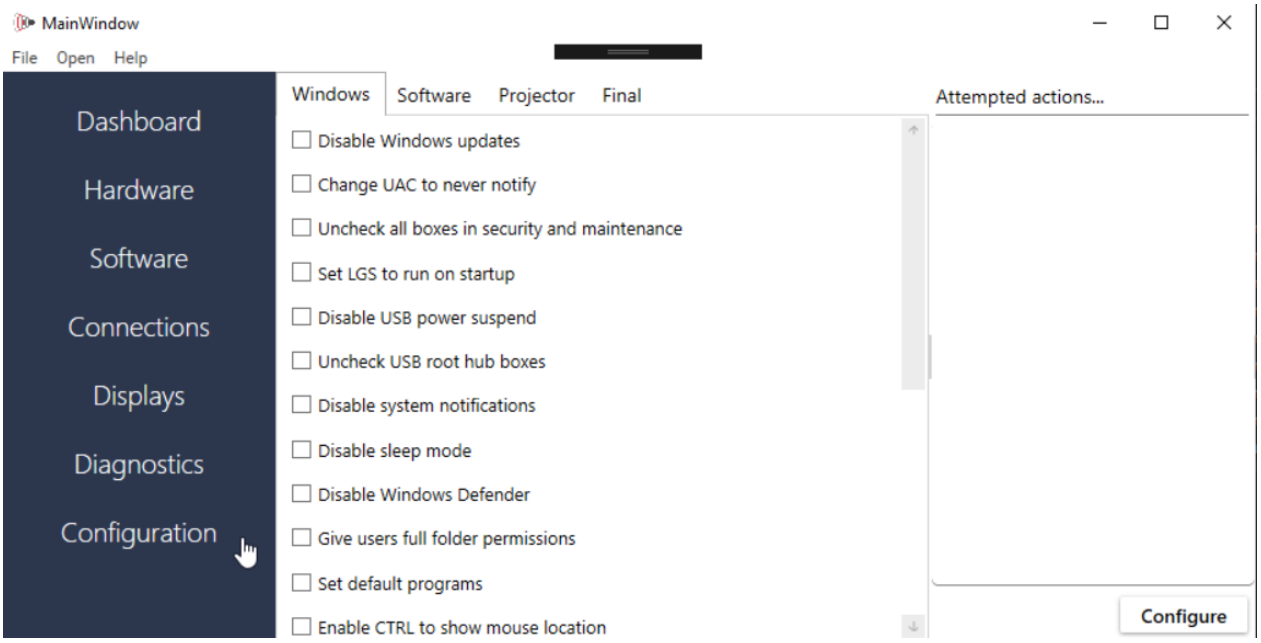

Figure 12: *Legacy Sonar Diagnostic Page*

Action Implementation

By looking at the Gannt Chart for the project timeline, the most time was reserved for Action Implementation. That is because this was the most challenging and time intensive part of developing Sonar. After all the pages were developed, the Configuration Page needed the functionality of making the changes described in the Integration Checklist. This was going to be quite challenging as it mostly required making changes to various Windows settings. This is difficult for two main reasons: first, Windows makes it difficult to programmatically change settings for security reasons, and second, there is no main directory of all settings that can be easily edited. The Windows Registry is the closest thing available to a global directory of all Windows settings, but it can be difficult to modify and there is not necessarily an editable registry key for every setting available in the Settings GUI. The registry will be used, when possible, but when the action cannot

be performed by registry edits other methods must be employed. One alternative method used to perform these actions is editing the necessary settings via Command Line or PowerShell. Automation scripts can be written within Visual Studio and deployed using either configuration management program. Another technique implemented to perform these actions is by using commands from within Visual Studio. Since Visual Studio is developed by Microsoft, they provide many commands, either natively or added via Reference, that can affect change on different aspects of the Windows OS. The final method, and by far the most difficult method, that was implemented to perform these actions was using Platform Invocation Services (PInvoke). This is a technique that can be used from within Visual Studio to reference unmanaged APIs used by Windows to access information. This method is the most difficult because there is a lot of overhead necessary to setup this approach. Another reason this method can be tricky is that said APIs are unmanaged, meaning that it complies directly to machine code with no intermediate languages (Basagalla, 2016).

As mentioned previously, the other main hurdle is that Windows intentionally limits the ways you can programmatically access these settings so they cannot be exploited by malware and other bad actors. This makes developing an application to configure Windows settings very difficult, but not impossible, since the need for programmatic settings modifications exists within the IT space already. After countless hours searching the web, workarounds can be found. The first step in overcoming this challenge is to make sure that the application is running as Administrator. This is easy to accomplish during development, it is as simple as running Visual Studio as Administrator while programming. However, when the application is packaged for installation on client

machines it will require the user to run the application as Administrator manually. Fortunately, there is setting that can be changed from within Visual Studio that will elevate privileges. In the Sonar app.manifest file the requested execution level must be changed to: "`<requestedExecutionLevel level="requireAdministrator" uiAccess="false"/>`" (Mulka, 2021, app.manifest). Making this change will elevate the User Account Control (UAC) and prompt the user to run the program as Administrator when it is launched. Without running as Administrator, a lot of the actions will fail when they are attempted to be configured.

Another key distinction to be made when developing these actions concerns the nomenclature of the Configuration Page. Actions begin as "Incomplete," once they've been performed, they become "Complete" but still must be verified before moving to the category of the same name. The important difference here is that the items in the "Verified" category have feedback validating their completion. When the "Configure" button is clicked the program performs the action it is told to do. That action can either be performed correctly or something might occur that prevents it from being completed. In the case of the latter, we have no feedback declaring to the program that something went wrong. It would be catastrophic to the program if it thinks it is performing actions, but they are being performed incorrectly or not at all. That is where the verification routines come in handy. When the "Verify" button is pressed a second routine is called that checks to ensure that the setting has been modified correctly. The same principle is used in electro-mechanical systems with motors. The controller will tell the motor to move a specified distance and the motor will do so, but the motor itself has no way of knowing that it has reached the destination. In this case a rotary encoder is added that effectively

“counts” how far the motor has traveled to ensure that it makes it to the desired position. In this hypothetical scenario the rotary encoder is the verification routine, and the motor corresponds to the configuration routine.

Now onto the fun part, actually performing these actions. To achieve this there is a custom class called ListViewAction, which Inherits its attributes from the ListViewItem Class. This is because the boxes seen on the Configuration Page are ListViews so they must be populated with ListViewItems. ListViewAction is a custom ListViewItem with additional methods so that it can be used in this project. The first major differentiator is that each ListViewAction has a checkbox associated with it, this is what allows the user to select the action they want. Additionally, each ListViewAction also has 3 custom properties: ActionToPerform, VerifyActionWasPerformed, and Status. Status is an integer Enum that holds a numerical value representing the actions completion status. For example, “Incomplete” corresponds to a 0 and “Verified” is represented with a 2. ActionToPerform and VerifyActionWasPerformed are both delegate functions. “A delegate is type-safe function pointer that can reference a method that has the same signature as that of the delegate” (Kanjilal, 2015). In essence, a delegate is a variable that holds a method. Delegates act as a middleman between the method to be called and the object that is calling that method. In this case, the object is the ListViewAction, the delegates are ActionToPerform and VerifyActionWasPerformed and the methods are functions that actually modify the Windows settings. The final component of ListViewAction is the constructor that allows for the creation of these actions. The constructor takes four parameters: the name to be displayed, the completion status of the action, the delegate pointing to the function that performs the action and the delegate

pointing to the function that verifies the completion of the action. All these constructors are initialized in a Class called ActionAssigner. The ActionAssigner Class, in full, is attached as Appendix D.

Now that all the actions have been created, they need to be added to the Configuration page. When the Configuration Page loads, it imports each action from the list created in the ActionAssigner Class, checks its status property and places it in the correct ListView according to its status. When the user desires to perform the action, they will select the checkbox for the desired action and press the “Configure” button. When this button is pressed it invokes the ActionToPerform delegate. Invoking is the process of executing the method called by the delegate. When the delegate is invoked, it performs the specified method on the associated thread (Microsoft, 2021). Since all these methods are functions, they will return a Boolean declaring whether the function was successful or not. If the function was successful, it will return true and get moved to the “Verify” section. The same process as described above applies to the actions in the “Verify” section when the “Verify” button is pressed. The only difference being that when this button is pressed it invokes the VerifyActionWasPerformed delegate rather than the ActionToPerform delegate.

Examples of Action Methods

After outlining the process of implementing these actions, this section will provide examples of a couple unique action methods. The purpose of this is to get a better understanding of how the program actually performs the actions contained within. These examples correspond to the techniques first introduced in the opening paragraph of the Action Implementation section.

The first, and most common, method for performing these actions is by using registry edits. The Windows Registry is an editable database for Windows Settings. The elements in this database are identified by a “key” which is akin to the file path and a “value” which is a name/data pair of information stored within the key. The process for making a registry edit from within Visual Studio is to first find the settings that needs to be changed in the registry. For this example, the action in question is to disable Windows updates. This setting can be found in the registry with a key of "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer". The value associated with the updates is “NoWindowsUpdate”. So, to disable this update the method will navigate to the key above and change the value to 0, which represents disablin g updates. This can be seen in the code block below:

```
Public Function DisableWindowsUpdates() As Boolean
        Try
            Dim key As String =
"HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Exp
lorer"
            Dim value As String = "NoWindowsUpdate"
            Dim data As Integer = 0 'disable

            My.Computer.Registry.SetValue(key, value, data)
            ConfigurationPage.AttemptedActions.Add("SUCCESS: Windows
updates disabled")
            Return True
        Catch ex As Exception
            ConfigurationPage.AttemptedActions.Add($"FAILURE: could not
disable Windows updates - {ErrorToString()}")
            Return False
        End Try
End Function
```

(Mulka, 2021, Sonar.vb, lines 85-98).

The operative line in the code above is: `“My.Computer.Registry.SetValue(key, value, data)”` (Mulka, 2021, Sonar.vb, line 91). This line executes as, navigate to the given key and set the given value to the desired data. This example is a good place to highlight how these configuration methods don’t provide any feedback on the outcome of the process.

After the SetValue command runs, the function returns true without verifying if updates were actually disabled. If the application encounters an error (handled exception) it will then return false but that is just to signify that the application was not even able to attempt this action due to an unforeseen condition. This is precisely why the verify method is necessary. The verify method for this action is quite similar as can be seen below:

```
Public Function VerifyWindowsUpdates() As Boolean
        Dim key As String =
"HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Exp
lorer"
        Dim value As String = "NoWindowsUpdate"

        If My.Computer.Registry.GetValue(key, value, Nothing) = 0 Then
            ConfigurationPage.AttemptedActions.Add("Pass: Windows updates
disabled")
            Return True
        Else
            ConfigurationPage.AttemptedActions.Add($"Fail: Windows
updates have not been disabled")
            Return False
        End If
End Function
```

(Mulka, 2021, Sonar.vb, lines 99-110).

The main difference here is instead of changing the value it reads the value and makes sure that it equals 0. If so, then the method returns true, if not, the method returns false.

The next strategy used to configure the actions is to use the Command Prompt or PowerShell. These Windows-native configuration management programs can be opened, fed arguments, and run all from within Visual Studio. The Command Prompt is quite helpful for developers as it gives you access to very specific low-level Windows kernel and OS settings that are not available in the standard GUIs. For this example, this method uses the Command Prompt to configure the systems IP address & subnet mask:

```
Public Function SetIP() As Boolean
        Try
            Dim command As String = "netsh Interface ipv4 Set address
name='LGS Network' Static 192.168.2.101 255.255.255.0"
```

```
            Dim p As Process = New Process()
            Dim pi As ProcessStartInfo = New ProcessStartInfo()
            pi.Arguments = " " + "/C" + " " + command
            pi.FileName = "cmd.exe"
            p.StartInfo = pi
            p.Start()

            ConfigurationPage.AttemptedActions.Add("SUCCESS: Network
information has been properly set")
            Return True
        Catch ex As Exception
            ConfigurationPage.AttemptedActions.Add($"FAILURE: Unable to
properly set network information - {ErrorToString()}")
            Return False
        End Try
End Function
```

(Mulka, 2021, Sonar.vb, lines 198-214).

The “command” variable is the string that would be typed into the Command Prompt and the remainder of the variables are standard for initializing the Command Prompt within Visual Studio. Calling p.Start() launches the Command Prompt with the arguments provided in the proceeding lines. This example is good to show how the verify method doesn’t always follow the same process as the configuration method. This can be seen by looking at the verify function that checks the IP and subnet mask:

```
Public Function VerifyIP() As Boolean
        Dim adapters As NetworkInterface() =
NetworkInterface.GetAllNetworkInterfaces()
        Dim IPdict As Dictionary(Of String, String) = New Dictionary(Of
String, String)

        For Each adapter As NetworkInterface In adapters
            Dim properties As IPInterfaceProperties =
adapter.GetIPProperties()
            Dim ip As String
            Dim firstnum As String

            ip = properties.UnicastAddresses(1).Address.ToString()
            firstnum = ip.Substring(0, ip.IndexOf("."))

            If Val(firstnum) > 0 And Not Val(firstnum) = 169 And Not
Val(firstnum) = 127 Then
                IPdict.Add(adapter.Name.ToString(), ip)
            End If
        Next

        If IPdict.TryGetValue(Constants.LGS_NETWORK, "192.168.2.101")
Then
            ConfigurationPage.AttemptedActions.Add("Pass: LGS Netork
information has been set properly")
            Return True
        Else
```

```
            ConfigurationPage.AttemptedActions.Add($"Fail: Could not
verify IP address has been set properly")
            Return False
        End If
End Function
```

(Mulka, 2021, Sonar.vb, lines 215-241).

It is obvious looking at the verify routine that it doesn't share many similarities with the configuration method. That is because the Command Prompt is great for setting values from within Visual Studio but not as good at retrieving values. Since verifying the IP address and subnet mask requires values to be fed back to the code another method needs to be implemented. The method above uses Platform Invocation to verify that the IP and subnets were set correctly.

Platform Invocation (PInvoke) is the final method used to construct these methods. "PInvoke is a technology that allows you to access structs, callbacks, and functions in unmanaged libraries from your managed code" (Microsoft, 2021). Languages such as C/C++ or Visual Basic, the language used in this project, are inherently managed but can be used in conjunction with unmanaged code. However, this must be done with extreme caution since the code is unmanaged it is prone to memory leaks with little to no feedback on what is causing it. The general premise for interacting with unmanaged code is to import the desired dynamic link library (DLL), construct a managed version of the necessary methods, and then marshal the unmanaged data into these managed functions. Marshaling is the process of transforming data types between managed and unmanaged code (Microsoft, 2021). This is necessary because managed code uses top level data typed such as strings and arrays, where unmanaged code is restricted to primitive data types since it compiles to machine code. Marshaling acts as a bridge between the memory representation and the accessible object available within

Visual Studio. An example of this process in Sonar can be found in the methods for setting the recommended display resolution. This method used some unmanaged functions that are specific to the User32.dll. The first step is to import this DLL, which can be done rather simply by calling "`<DllImport("user32.dll")>`" (Mulka, 2021, Sonar.vb, line 725). This DLL gives the application access to the EnumDisplaySettings() function which will be instrumental in setting the correct resolution for all displays on the system. This function can be defined in VB.NET as:

```
Public Function EnumDisplaySettings(
     <MarshalAs(UnmanagedType.LPTStr)> ByVal lpszDeviceName As String,
     <MarshalAs(UnmanagedType.U4)> ByVal iModeNum As Integer,
     <Runtime.InteropServices.In, Out> ByRef lpDevMode As DEVMODE) As
<MarshalAs(UnmanagedType.Bool)> Boolean
End Function
```

(Mulka, 2021, Sonar.vb, lines 726-730).

Here it is shown how the unmanaged data types are being marshaled into the equivalent higher level data types. This function is the reason PInvoke must be used. It would be possible to programmatically change the display resolution from the registry, however, there would be no guarantee that you have changed the resolution for all display devices on the system. This function enumerates every connected display device on the system, so Visual Studio can build a list of objects that refer to each of the display devices. Once there is an object to refer to that display device the settings can be easily changed. Looking at the list of data types, all are pretty standard with the exception of DEVMODE. That is because DEVMODE is a custom Structure that must be defined within Visual Studio. The DEVMODE structure is a collection of memory that encapsulates information about a display device and can be defined as:

```
<StructLayout(LayoutKind.Sequential, CharSet:=CharSet.Ansi)>
Public Structure DEVMODE
    <MarshalAs(UnmanagedType.ByValTStr, SizeConst:=32)>
    Public dmDeviceName As String
    <MarshalAs(UnmanagedType.U2)>
    Public dmSpecVersion As UInt16
```

```
    <MarshalAs(UnmanagedType.U2)>
    Public dmDriverVersion As UInt16
    <MarshalAs(UnmanagedType.U2)>
    Public dmSize As UInt16
    <MarshalAs(UnmanagedType.U2)>
    Public dmDriverExtra As UInt16
    <MarshalAs(UnmanagedType.U4)>
    Public dmFields As UInt32
    Public dmPosition As POINTL
    <MarshalAs(UnmanagedType.U4)>
    Public dmDisplayOrientation As UInt32
    <MarshalAs(UnmanagedType.U4)>
    Public dmDisplayFixedOutput As UInt32
    <MarshalAs(UnmanagedType.I2)>
    Public dmColor As Int16
    <MarshalAs(UnmanagedType.I2)>
    Public dmDuplex As Int16
    <MarshalAs(UnmanagedType.I2)>
    Public dmYResolution As Int16
    <MarshalAs(UnmanagedType.I2)>
    Public dmTTOption As Int16
    <MarshalAs(UnmanagedType.I2)>
    Public dmCollate As Int16
    <MarshalAs(UnmanagedType.ByValTStr, SizeConst:=32)>
    Public dmFormName As String
    <MarshalAs(UnmanagedType.U2)>
    Public dmLogPixels As UInt16
    <MarshalAs(UnmanagedType.U4)>
    Public dmBitsPerPel As UInt32
    <MarshalAs(UnmanagedType.U4)>
    Public dmPelsWidth As UInt32
    <MarshalAs(UnmanagedType.U4)>
    Public dmPelsHeight As UInt32
    <MarshalAs(UnmanagedType.U4)>
    Public dmDisplayFlags As UInt32
    <MarshalAs(UnmanagedType.U4)>
    Public dmDisplayFrequency As UInt32
    <MarshalAs(UnmanagedType.U4)>
    Public dmICMMethod As UInt32
    <MarshalAs(UnmanagedType.U4)>
    Public dmICMIntent As UInt32
    <MarshalAs(UnmanagedType.U4)>
    Public dmMediaType As UInt32
    <MarshalAs(UnmanagedType.U4)>
    Public dmDitherType As UInt32
    <MarshalAs(UnmanagedType.U4)>
    Public dmReserved1 As UInt32
    <MarshalAs(UnmanagedType.U4)>
    Public dmReserved2 As UInt32
    <MarshalAs(UnmanagedType.U4)>
    Public dmPanningWidth As UInt32
    <MarshalAs(UnmanagedType.U4)>
    Public dmPanningHeight As UInt32
End Structure
```

(Mulka, 2021, DEVMODE Structure.vb).

Within this structure there is another structure called POINTL but that is not relevant to the scope of this thesis. Of all these members included in the structure only a handful are pertinent to the task at hand, chiefly among them being dmPelsWidth and dmPelsHeight.

These 2 members hold the width and height, in pixels, of the given display. There is one additional user32.dll function necessary to change the display resolution and that is the ChangeDisplaySettings() function:

```
Public Function ChangeDisplaySettings(
          <Runtime.InteropServices.In, Out> ByRef lpDevMode As DEVMODE,
          <MarshalAs(UnmanagedType.U4)> ByVal dwflags As Integer) As
<MarshalAs(UnmanagedType.I4)> Integer
End Function
```

(Mulka, 2021, Sonar.vb, lines 751-754).

This function also makes use of the same DEVMODE structure that EnumDisplaySettings() uses. This function only takes two arguments lpDevMode and dwflags. lpDevMode is the DEVMODE structure previously looked at and dwflags is not relevant for this application so it will be set to zero. At this point it becomes evident why this method is the last resort for programming these actions. All these steps are required before even beginning to develop the configuration action. Now that all the overhead has been taken care of the configuration function can be written:

```
Public Function RecommendedResolution() As Boolean
        Try
            Dim newWidth As Integer = 1920
            Dim newHeight As Integer = 1080

            Dim activeDisplays As New List(Of String)
            activeDisplays = GetDisplayDevices()

            For Each display As String In activeDisplays
                Dim result As Boolean = ChangeDisplayResolution(newWidth,
newHeight, display)
                If result Then
                    'Do nothing
                Else
                    Throw New Exception("Please refer to log for error")
                End If
            Next
            ConfigurationPage.AttemptedActions.Add($"SUCCESS: Display
resoution has been successfully changed to {newWidth}x{newHeight}")
            Return True
        Catch ex As Exception
            ConfigurationPage.AttemptedActions.Add($"FAILURE: Could not
change display resolution - {ErrorToString()}")
            Return False
        End Try
End Function
```

(Mulka, 2021, Sonar.vb, lines 282-307).

The first step in executing this code is to establish the variables, in this case it is the desired width and height of the screen resolution. Next, the GetDisplayDevices() function is called which returns a list of device names that correspond to all of the display devices on the system. The code then loops through all displays and calls the ChangeDisplayResolution() function. This function is a wrapper for the ChangeDisplaySettings() function discussed earlier, specifically for the purpose of changing the resolution. This function takes the new desired height and width as well as the name for the display device to apply these changes to. The function then accesses the DEVMODE structure for the display device it was passed and changes the dmPelsWidth and dmPelsHeight parameter discussed previously. Once the values are changed it applies the modified DEVMODE structure to the given display device and goes onto the next display device.

IV. RESULTS

As for the lofty goals outlined in the Introduction, they were all attempted with varying levels of completion. After the completion of phase I, it was clear that this application would have a significant impact on the time required to integrate a LightGuide System. However, no data was able to be collected to validate this. In its current state the application provides plenty of information to aid in troubleshooting such as: the system hardware and software statistics, current LightGuide Application information, and changes any changes to PnP devices on the system. Sonar was successfully migrated from WPF to WinForms and many of the original features were able to be salvaged with a fair amount of refactoring to compensate for the change in graphical interface. Sonar

also currently does provide a generated report, referred to as “Integration Document”, but it is not formatted to resemble the currently used Integration Checklist. The application also does warn the user of actions that may not be completed as defined by the checklist, however, it does not yet provide the user with instructions on how to complete the task manually. Another goal that has been attempted, but not fully completed is the UI. The application makes use of sophisticated custom controls to provide the feel of a professional application but does not have final graphics or an overall scheme. All the artwork was created in Microsoft Paint as placeholder art until time could be dedicated to finalizing the graphics. It also currently uses the default WinForms color scheme. In the future the application will make use of a color scheme more consistent with LightGuide branding requirements. Additionally, in the later phases of development the program will be able to configure a PC completely from scratch without the need for applying an image. Progress will continue to be made on Sonar in the future to bring all these features to the software. Sonar has the potential to be a huge source of value for the company, both as a configuration and as a diagnostic tool. Finally, this project proved to be an incredibly beneficial tool to get familiar with the LightGuide development process and increase the author’s ability to develop custom code applications.

VI. REFERENCES

Basagalla, R. (2016, April 29). *The Difference Between Managed and Unmanaged Code in .Net*. Retrieved from C# Corner: https://www.c-sharpcorner.com/blogs/difference-between-managed-and-unmanaged-code-in-net

Kanjilal, J. (2015, October 23). *How to work with delegates in C#*. Retrieved from InfoWorld: https://www.infoworld.com/article/2996770/how-to-work-with-delegates-in-csharp.html

Microsoft. (2021). *Control.Invoke Method*. Retrieved from Microsoft Docs: https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.control.invoke?view=windowsdesktop-6.0

Microsoft. (2021). *Module Statement*. Retrieved 2021, from Microsoft Docs: https://docs.microsoft.com/en-us/dotnet/visual-basic/language-reference/statements/module-statement

Microsoft. (2021). *Platform Invoke (P/Invoke)*. Retrieved from Microsoft Docs: https://docs.microsoft.com/en-us/dotnet/standard/native-interop/pinvoke

Microsoft. (2021). *Type Marshaling*. Retrieved from Microsoft Docs: https://docs.microsoft.com/en-us/dotnet/standard/native-interop/type-marshaling

Mulka, D. (2021). Sonar Source Code. *Version 2.0*.

VII. APPENDICES AND SUPPLEMENTAL MATERIALS

Appendix A - Integration Checklist

New PC Configuration Checklist

Date: 11/12/2021

Starting from a fresh Windows 10 install on **Dell 7090**

Wiki Date:_____ LGS Version #:_____

Dongle Serial #:_____

#	Task	Initials	Image
1.0	**Hardware**	XXXXXXXX	XXXXXXXX
1.1	Install GPU		
1.2	Install Serial Cards (if applicable)		
1.3	Install USB Cards (if applicable)		
1.4	Install Ethernet Cards		
1.5	Install Internal motherboard adapter (if applicable) (5820 already has) (7090 needs it)		
2.0	**Windows 10 Setup**	XXXXXXXX	XXXXXXXX
2.1	Location/Keyboard settings: United States/US		
2.2	Click "I don't have internet" > Continue with limited setup > Accept		
2.3	Username: LGS-User		
2.4	Leave password blank		
2.5	No Activity History		
2.6	Decline Digtal Assistant		
2.7	Disable all privacy options		
2.8	No Product Key		
3.0	**Imaging Setup**	XXXXXXXX	XXXXXXXX
3.1	Run Command Prompt as an Administrator		
3.2	Type in: *bcdedit /set {current} safeboot minimal* then press enter		
3.3	Restart the computer and enter UEFI/BIOS setup		
3.4	System config > SATA Operation > AHCI		
4.0	**BIOS Settings**	XXXXXXXX	XXXXXXXX
4.1	Press F12 during boot up	XXXXXXXX	XXXXXXXX
4.2	Bios Settings > View All	XXXXXXXX	XXXXXXXX
4.3	Storage > SMART Reporting > Enable SMART Reporting > (Toggle) ON		
4.4	Power > USB Wake Support > Enable USB Wake Support > (Toggle) OFF		
4.5	Power > AC Behavior > AC Behavior > Power On		
4.6	Power > Active State Power Management > Active State Power Management > Disable		
4.7	Power > Block Sleep > Block Sleep > (Toggle) OFF		
4.8	Power > Deep Sleep Control > Deep Sleep Control > Disable		

4.9	Update, Recovery > UEFI Capsule Firmware Updates > Enable UEFI Capsule Firmware Updates > (Toggle) OFF		
4.10	Performance > Intel Speedstep > (Toggle) OFF		
4.11	Performance > C-States Control > (Toggle) OFF		
5.0	**Initial Settings**	XXXXXXXX	XXXXXXXX
5.1	Connect to internet via ethernet	XXXXXXXX	XXXXXXXX
5.2	Search "Date" in the Start Menu > Date & Time Settings > Set Time Automatically > (Toggle) ON		
5.3	Search "Update" in the Start Menu > Check for Updates > Click "Update" (if available)		
5.4	Advanced Options > turn on Receive update for other products when you update		XXXXXXXX
5.5	Delivery Optimization > turn off		XXXXXXXX
5.6	Search "Services" in the Start Menu> (Double Click) Windows Update > Stop > Startup Type: Disable (Drop Down) > OK		
5.7	Make an "OPS Solutions" folder in C:\ProgramFiles		XXXXXXXX
5.8	Copy installation files into this folder		XXXXXXXX
5.9	Disconnect from Internet (Unplug Ethernet)	XXXXXXXX	XXXXXXXX
6.0	**Control Panel**	XXXXXXXX	XXXXXXXX
6.1	Search "Control Panel" in the Start Menu	XXXXXXXX	XXXXXXXX
6.2	Go into System & Security	XXXXXXXX	XXXXXXXX
6.3	Security & Maintenance Menu	XXXXXXXX	XXXXXXXX
6.4	Security > User Account Control > Change Settings > Never Notify > OK		XXXXXXXX
6.5	Change Security & Maintenance settings > Uncheck all settings > OK		
6.6	Windows Defender Firewall Menu	XXXXXXXX	XXXXXXXX
6.7	Turn windows defender firewall on or off > turn off public & private firewall > OK		XXXXXXXX
6.8	Power Options Menu	XXXXXXXX	XXXXXXXX
6.9	Change plan settings > Turn off the display > Never (Drop Down) > Save Changes		XXXXXXXX
6.10	Change advanced power settings > USB settings > USB selective suspend: Disable (Drop Down)		XXXXXXXX
6.11	PCI Express > Link State Power Management > Turn Off (Drop Down)		XXXXXXXX
6.12	Processor power management > set minimum processor state to 100% > OK		XXXXXXXX
6.13	Go into Network & Internet	XXXXXXXX	XXXXXXXX
6.14	Network & Sharing Center Menu	XXXXXXXX	XXXXXXXX
6.15	Change adapter settings > Rename networks (Right Click)		
6.16	Change motherboard port to "Plant Network" and PCIe port to "LGS Network"		
6.17	Right click LGS Network > Properties > double click IPV4 > use the following IP		
6.18	Change IP to "192.168.2.101"		
6.19	Change Subnet to "255.255.255.0" > OK		

6.20	Go into Hardware and Sound	XXXXXXXX	XXXXXXXX
6.21	Beneath Devices & Printers click on mouse		XXXXXXXX
6.22	Pointers > Scheme > "Windows Inverted (XL) (system scheme)"		XXXXXXXX
6.23	Pointer Options > turn on "show location of pointer when I press CTRL"		XXXXXXXX
6.24	Search "Keyboard" in the Start Menu > set repeat delay to long		XXXXXXXX
7.0	**Graphics**	XXXXXXXX	XXXXXXXX
7.1	Open Nvidia Control Panel from the System Tray		
7.2	Work Station > View System Topology		
7.3	System > Quadro > Display Port (#) > EDID		
7.4	Browse > C:\Program Files\OPS Solutions\Install Files\2-Hardware Files\Graphics\Nvidia PCIe Video Cards\EDIDs		
7.5	Select the projector model that is being used on the install		
7.6	Check all connector that will have projectors plugged in (Usually DisplayPort (1),(3),(4)		
7.7	Check the Warning		
7.8	Load EDID		
8.0	**Windows Settings**	XXXXXXXX	XXXXXXXX
8.1	Open the Windows Settings	XXXXXXXX	XXXXXXXX
8.2	Go into System	XXXXXXXX	XXXXXXXX
8.3	Display Menu	XXXXXXXX	XXXXXXXX
8.4	Scale and Layout > Change the size of text… > 100% (Drop Down)		
8.5	Display resolution > … (Recommended) (Drop Down)		
8.6	Multiple Displays > Select unused displays > Disconnect (Drop Down)		
8.7	Repeat last three steps for all displays		
8.8	Sound Menu	XXXXXXXX	XXXXXXXX
8.9	Output > Set output device to speaker (internal/external)		
8.10	Notifications & Actions Menu	XXXXXXXX	XXXXXXXX
8.11	Edit your quick actions > unpin all tiles		XXXXXXXX
8.12	Notifications > Get notifications from apps and other senders > (Toggle) OFF		XXXXXXXX
8.13	Focus Assist Menu > Turn off all settings in this menu		
8.14	Power & Sleep Menu	XXXXXXXX	XXXXXXXX
8.15	Set all drop downs to never		XXXXXXXX
8.16	Go into Devices	XXXXXXXX	XXXXXXXX
8.17	Printers & Scanners Menu		XXXXXXXX
8.18	Remove all printers (Right Click)		XXXXXXXX
8.19	Autoplay Menu	XXXXXXXX	XXXXXXXX
8.20	Use AutoPlay for all media and devices > (Toggle) ON		XXXXXXXX
8.21	Choose AutoPlay defaults > Removeable drive / Memory card > Open folder to… (Drop Down)		XXXXXXXX
8.22	Go into Personalization	XXXXXXXX	XXXXXXXX
8.23	Background Menu	XXXXXXXX	XXXXXXXX

8.24	Browse > Installation Files > 1a-LightGuide > Logos & Wallpaper > Classic Red > Rays Red		XXXXXXXX
8.25	Colors Menu	XXXXXXXX	XXXXXXXX
8.26	Choose your color > Dark (Drop Down)		XXXXXXXX
8.27	Transparency > (Toggle) OFF		XXXXXXXX
8.28	Choose your accent color > Red		XXXXXXXX
8.29	Lock Screen Menu	XXXXXXXX	XXXXXXXX
8.30	Background (Drop Down) > Picture > Browse > LGS logo		XXXXXXXX
8.31	Get fun facts, tips, and more from Windows and Cortana on your lock screen > (Toggle) OFF		XXXXXXXX
8.32	Choose one / which app(s) show … on the lock screen > select all > None (Drop Down)		XXXXXXXX
8.33	Start Menu	XXXXXXXX	XXXXXXXX
8.34	Show app list in Start menu > leave on > (Toggle other settings) OFF		XXXXXXXX
8.35	Taskbar Menu	XXXXXXXX	XXXXXXXX
8.36	Lock the taskbar > (Toggle) ON		XXXXXXXX
8.37	Select which icons appear on the taskbar > Always show all icons… > (Toggle) ON		XXXXXXXX
8.38	Show taskbar on all displays > (Toggle) OFF		XXXXXXXX
8.39	Show contacts on the taksbar > (Toggle) OFF		XXXXXXXX
8.40	Go into Apps	XXXXXXXX	XXXXXXXX
8.41	Apps & Feature Menu	XXXXXXXX	XXXXXXXX
8.42	Choose where to get apps > Anywhere (Drop Down)		XXXXXXXX
8.43	Remove all unnecessary apps from PC (anything that isn't critical to the job)		XXXXXXXX
8.44	Offline Maps Menu > disable auto update maps		XXXXXXXX
8.45	Startup Menu	XXXXXXXX	XXXXXXXX
8.46	Disable update utilities, audio programs, logitech, onedrive		
8.47	Go into Accounts > Set photo to LGS_Rays1080		XXXXXXXX
8.48	Go into Gaming > Game Bar > (Toggle) OFF (Repeat for Game Mode)		XXXXXXXX
8.49	Go into Ease of Access	XXXXXXXX	XXXXXXXX
8.50	Display Menu > Disable Show animations, transparency, auto hide scroll bars		XXXXXXXX
8.51	Go into Privacy	XXXXXXXX	XXXXXXXX
8.52	General Menu > (Toggle) OFF (Everything)		XXXXXXXX
8.53	Speech Menu > Online speech recognition > (Toggle) OFF		XXXXXXXX
8.54	Inking & Typing Menu > Getting to know you > (Toggle) OFF		XXXXXXXX
8.55	Diagnostic & Feedback Menu	XXXXXXXX	XXXXXXXX
8.56	(Select) Requred diagnostic data		XXXXXXXX
8.57	Disable other settings		XXXXXXXX
8.58	Feedback frequency > Never (Drop down)		XXXXXXXX
8.59	Activity History Menu > Diselect everything		XXXXXXXX

8.60	Camera Menu > (Toggle) ON		XXXXXXXX
8.61	Microphone Menu > (Toggle) ON		XXXXXXXX
8.62	Notifications Menu > (Toggle) OFF		XXXXXXXX
8.63	Other Devices Menu > (Toggle) ON		XXXXXXXX
8.64	Background Apps Menu > (Toggle) OFF		XXXXXXXX
8.65	Go into Update & Security	XXXXXXXX	XXXXXXXX
8.66	Windows Update Menu > Advanced options > (Toggle) OFF all options		XXXXXXXX
8.67	Delivery Optimization Menu > (Toggle) OFF		XXXXXXXX
8.68	Windows Security Menu	XXXXXXXX	XXXXXXXX
8.69	Virus & Threat Protection > Manage Settings > Disable all options		XXXXXXXX
8.70	App & Browser Control > Turn off all options		XXXXXXXX
9.0	**Install Programs**	XXXXXXXX	XXXXXXXX
	LGS	XXXXXXXX	XXXXXXXX
9.1	go to "C:\Program Files\OPS Solutions\Installation Files\"		XXXXXXXX
9.2	Choose the most recent version from "Current Production Versions" folder		XXXXXXXX
9.3	Unzip and run "Setup.exe"		XXXXXXXX
9.4	Navigate to "C:\Program Files\"		XXXXXXXX
9.5	Right click "OPS Solutions" > Properties > Security		XXXXXXXX
9.6	click edit > select "Users" > Check "Allow" next to "Full Control"		XXXXXXXX
9.7	Open "LightGuide 64 No Login" > Close LightGuide		
9.8	Go to "C:\Program Files\OPS Solutions\Installation Files\LGS Install\Documentation"		XXXXXXXX
9.9	Run the wiki installer		XXXXXXXX
	Drivers	XXXXXXXX	XXXXXXXX
9.10	Install necessary drivers for Realsense, Kinect, Orbbec, serial cards		
	Other Programs	XXXXXXXX	XXXXXXXX
9.11	Install Chrome, VLC, Notepad++		XXXXXXXX
9.12	Go to Windows Settings > Apps > Default Apps	XXXXXXXX	XXXXXXXX
9.13	Music player > VLC		
9.14	Photo Viewer > Paint		
9.15	Video Player > VLC		
9.16	Web Browser > Chrome		
9.17	Make Chrome default apps for .pdf files		XXXXXXXX
9.18	If system has Cognex, Gateways or webcams download necessary software		
10.0	**Other Settings**	XXXXXXXX	XXXXXXXX
10.1	Search "Device Manager" in the start menu > Right click all USB items in "Human Interface Devices"		
10.2	Properties > Power Management Tab > Diselect "Allow computer to turn off device" > OK		
10.3	Repeat for devices in "USB Controllers", "Orbbec", "Realsense"		
10.4	Install Logitech Unifying Software and pair deviced to keyboard dongle		

10.5	Make notepad++ default for .csv, .txt, .xml files	XXXXXXXX	XXXXXXXX
10.6	Run notepad++ as admin		
10.7	Open settings in ribbon		
10.8	Prefrences		
10.9	File association		
10.10	Select Notepad		
10.11	Select .txt > Press "->"		
10.12	Select Customize		
10.13	type in ".csv" > Press "->"		
10.14	type in ".xml" > Press "->"		
10.15	Search "Group Policy" in the Start Menu > Computer Config > Admin Templates > Windows Components > Search		XXXXXXXX
10.16	Double click "Allow Cortana" > Select Disable > OK		XXXXXXXX
10.17	Windows Components > Windows Update > Double click "Configure Auto Updates" > Disabled		XXXXXXXX
10.18	Right click taskbar > Disable show task view, show people, show Ink Workspace		XXXXXXXX
10.19	Right click taskbar > Search > change to show search icon		XXXXXXXX
10.20	Unpin all programs from taskbar except Chrome, File Explorer, Control Panel, LGS		XXXXXXXX
10.21	Open File Explorer > View tab > enable check boxes, extensions, hidden items		XXXXXXXX
10.22	View > Options > General > disable "show recently/frequently used"		XXXXXXXX
10.23	Set Open File Explorer to this PC > Clear > apply > Ok		
10.24	Right click on OPS solutions & LGS folders > Pin to quick access		XXXXXXXX
10.25	Open chrome > wiki.lightguidesys.com > Favorite the website > edit to "LGS Help"		XXXXXXXX
10.26	Enable show bookmarks bar		XXXXXXXX
10.27	Set hompage to wiki.lightguidesys.com		XXXXXXXX
10.28	Add additional bookmarks for projector IP addresses, DMX converter box IP		
10.29	Remove all programs off desktop except chrome, recycle bin, LGS, sonar, LGS		
10.30	Run sonar > copy sonar snapshot > save to project folder under LGS Backup		
10.31	Search "System Restore Point" > Create > Name it "In House Integration"		
10.32	PRJ to projector tab		
10.33	Canvas Screen		
10.34	Create a Test program in LGS to make sure all devices are functioning		

Appendix B - SystemReport.JSON Excerpt (Lines 1-116, 188-333)


```
{
  "LogTime": "2021-11-10T13:28:06",
  "InstalledProgramsList": [
    "Visual Studio Community 2019",
    "3uTools",
    "Arduino",
    "Audacity 2.3.3",
    "Citrix Workspace 1909",
    "Google Chrome",
    "Microsoft Edge",
    "Microsoft Edge Update",
    "Microsoft Edge WebView2 Runtime",
    "MiniTool ShadowMaker PW Edition",
    "Notepad++ (32-bit x86)",
    "OBS Studio",
    "P&E Device Drivers",
    "Adobe Photoshop CC 2019",
    "Razer Synapse",
    "Steam",
    "Lenovo Vantage Service",
    "Microsoft Visual C++ 2013 Redistributable (x64) - 12.0.30501",
    "vs_FileTracker_Singleton",
    "Windows SDK Desktop Tools arm64",
    "Windows IoT Extension SDK",
    "windows_toolscorepkg",
    "vs_minshellmsires",
    "WinRT Intellisense UAP - en-us",
    "Epic Online Services",
    "Microsoft Windows Desktop Runtime - 5.0.11 (x86)",
    "Windows Simulator - ENU",
    "UE4 Prerequisites (x64)",
    "Windows SDK for Windows Store Apps DirectX x86 Remote",
    "vs_filehandler_amd64",
    "icecap_collection_neutral",
    "Microsoft .NET CoreRuntime SDK",
    "Visual C++ Library CRT Desktop Appx Package",
    "WinRT Intellisense PPI - en-us",
    "Windows SDK Desktop Libs x64",
    "Microsoft .NET Framework 4.7.2 Targeting Pack",
    "Citrix Workspace (HDX Flash Redirection)",
    "Visual C++ Library CRT Appx Package",
    "Windows SDK Desktop Tools x86",
    "Windows SDK for Windows Store Apps Libs",
  ],
  "PreviousLogPath": null,
  "SystemSoftware": null,
  "SystemHardware": {
    "ComputerInfo": {
      "TotalPhysicalMemory": 17039765504,
      "AvailablePhysicalMemory": 7034081280,
      "TotalVirtualMemory": 4294836224,
      "AvailableVirtualMemory": 3846901760,
      "InstalledUICulture": "en-US",
      "OSFullName": "Microsoft Windows 10 Home",
```

```
    "OSPlatform": "Win32NT",
    "OSVersion": "6.2.9200.0"
  },
  "Processor": {
    "Properties": {
      "Caption": "Intel64 Family 6 Model 142 Stepping 10",
      "Description": "Intel64 Family 6 Model 142 Stepping 10",
      "InstallDate": null,
      "Name": "Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz",
      "Status": "OK",
      "Availability": 3,
      "ConfigManagerErrorCode": null,
      "ConfigManagerUserConfig": null,
      "CreationClassName": "Win32_Processor",
      "DeviceID": "CPU0",
      "ErrorCleared": null,
      "ErrorDescription": null,
      "LastErrorCode": null,
      "PNPDeviceID": null,
      "PowerManagementCapabilities": null,
      "PowerManagementSupported": false,
      "StatusInfo": 3,
      "SystemCreationClassName": "Win32_ComputerSystem",
      "SystemName": "DOUGS-PC",
      "AddressWidth": 64,
      "CurrentClockSpeed": 1801,
      "DataWidth": 64,
      "Family": 205,
      "LoadPercentage": 20,
      "MaxClockSpeed": 1801,
      "OtherFamilyDescription": null,
      "Role": "CPU",
      "Stepping": null,
      "UniqueId": null,
      "UpgradeMethod": 51,
      "Architecture": 9,
      "AssetTag": "To Be Filled By O.E.M.",
      "Characteristics": 252,
      "CpuStatus": 1,
      "CurrentVoltage": 9,
      "ExtClock": 100,
      "L2CacheSize": 1024,
      "L2CacheSpeed": null,
      "L3CacheSize": 6144,
      "L3CacheSpeed": 0,
      "Level": 6,
      "Manufacturer": "GenuineIntel",
      "NumberOfCores": 4,
      "NumberOfEnabledCore": 4,
      "NumberOfLogicalProcessors": 8,
      "PartNumber": "To Be Filled By O.E.M.",
      "ProcessorId": "BFEBFBFF000806EA",
      "ProcessorType": 3,
      "Revision": null,
      "SecondLevelAddressTranslationExtensions": true,
      "SerialNumber": "To Be Filled By O.E.M.",
      "SocketDesignation": "U3E1",
```

```
        "ThreadCount": 8,
        "Version": "",
        "VirtualizationFirmwareEnabled": true,
        "VMMonitorModeExtensions": true,
        "VoltageCaps": null,
        "PSComputerName": null
      },
      "Site": null,
      "Container": null
    },
    "OperatingSystem": {
      "Properties": {
        "Caption": "Microsoft Windows 10 Home",
        "Description": "",
        "InstallDate": "2021-03-27T23:29:57-04:00",
        "Name": "Microsoft Windows 10
Home|C:\\WINDOWS|\\Device\\Harddisk1\\Partition3",
        "Status": "OK",
        "CreationClassName": "Win32_OperatingSystem",
        "CSCreationClassName": "Win32_ComputerSystem",
        "CSName": "DOUGS-PC",
        "CurrentTimeZone": -300,
        "Distributed": false,
        "FreePhysicalMemory": 6795220,
        "FreeSpaceInPagingFiles": 2322856,
        "FreeVirtualMemory": 4841052,
        "LastBootUpTime": "2021-11-05T19:46:43.501722-04:00",
        "LocalDateTime": "2021-11-10T13:28:09.436-05:00",
        "MaxNumberOfProcesses": 4294967295,
        "MaxProcessMemorySize": 137438953344,
        "NumberOfLicensedUsers": 0,
        "NumberOfProcesses": 282,
        "NumberOfUsers": 2,
        "OSType": 18,
        "OtherTypeDescription": null,
        "SizeStoredInPagingFiles": 2714864,
        "TotalSwapSpaceSize": null,
        "TotalVirtualMemorySize": 19355260,
        "TotalVisibleMemorySize": 16640396,
        "Version": "10.0.19042",
        "BootDevice": "\\Device\\HarddiskVolume5",
        "BuildNumber": "19042",
        "BuildType": "Multiprocessor Free",
        "CodeSet": "1252",
        "CountryCode": "1",
        "CSDVersion": null,
        "DataExecutionPrevention_32BitApplications": true,
        "DataExecutionPrevention_Available": true,
        "DataExecutionPrevention_Drivers": true,
        "DataExecutionPrevention_SupportPolicy": 2,
        "Debug": false,
        "EncryptionLevel": 256,
        "ForegroundApplicationBoost": 2,
        "LargeSystemCache": null,
        "Locale": "0409",
        "Manufacturer": "Microsoft Corporation",
        "MUILanguages": [
```

```
    "en-US"
  ],
  "OperatingSystemSKU": 101,
  "Organization": "",
  "OSArchitecture": "64-bit",
  "OSLanguage": 1033,
  "OSProductSuite": 768,
  "PAEEnabled": null,
  "PlusProductID": null,
  "PlusVersionNumber": null,
  "PortableOperatingSystem": false,
  "Primary": true,
  "ProductType": 1,
  "RegisteredUser": "dougmulka@gmail.com",
  "SerialNumber": "00325-96515-00106-AAOEM",
  "ServicePackMajorVersion": 0,
  "ServicePackMinorVersion": 0,
  "SuiteMask": 784,
  "SystemDevice": "\\Device\\HarddiskVolume7",
  "SystemDirectory": "C:\\WINDOWS\\system32",
  "SystemDrive": "C:",
  "WindowsDirectory": "C:\\WINDOWS",
  "PSComputerName": null
},
```

Appendix C - Generated Integration Document

```
Integration Document_20211122.txt
Integrated by: Douglas Mulka
11/22/2021 11:01 PM

Verified Actions:
1. Set time automatically
2. Disable Windows updates
3. Uncheck all boxes in security and maintenance
4. Rename networks to Plant & LGS Network
5. Change IP Address & Subnet to defaults
6. Set text scaling to 100%
7. Set recommended display resolution
8. Disconnect unused displays
9. Set output device to speaker
10. Turn off all focus assist settings
11. Disable startup apps
12. Install Realsense, Kinect, Orbbec and Serial Card drivers
13. Set default apps
14. Disable Window's ability to turn off devices
15. Install Logitech Unifying Software
16. Configure Notepad++
17. Open file explorer to: This PC
18. Remove programs from desktop
19. Create system restore point
20. Generate Sonar System Report

Incomplete Actions:
1. Check for available updates
2. Set EDIDs
```

Appendix D - ActionAssigner.vb Class

```
Public Class ActionAssigner
      Public Shared Actions As New List(Of ListViewAction) From {

            New ListViewAction("Set time automatically", 0, AddressOf
SetTimeAuto, AddressOf VerifyTimeAuto),

            New ListViewAction("Check for available updates", -1),

            New ListViewAction("Disable Windows updates", 0, AddressOf
DisableWindowsUpdates, AddressOf VerifyWindowsUpdates),
            New ListViewAction("Uncheck all boxes in security and
maintenance", 0, AddressOf UncheckSAndM, AddressOf VerifySAndM),

            New ListViewAction("Rename networks to Plant & LGS
Network", 0, AddressOf RenameNetworks, AddressOf VerifyNetworkNames),

            New ListViewAction("Change IP Address & Subnet to
defaults", 0, AddressOf SetIP, AddressOf VerifyIP),

            New ListViewAction("Set EDIDs", -1),

            New ListViewAction("Set text scaling to 100%", 0, AddressOf
SetScaling, AddressOf VerifyScaling),

            New ListViewAction("Set recommended display resolution", 0,
AddressOf RecommendedResolution, AddressOf VerifyResolution),

            New ListViewAction("Disconnect unused displays", 0,
AddressOf DisconnectDisplays, AddressOf VerifyDisconnectDisplays),

            New ListViewAction("Set output device to speaker", 0,
AddressOf SetSpeaker, AddressOf VerifySpeaker),

            New ListViewAction("Turn off all focus assist settings", 0,
AddressOf DisableFocus, AddressOf VerifyDisableFocus),

            New ListViewAction("Disable startup apps", 0, AddressOf
DisableStartup, AddressOf VerifyDisableStartup),

            New ListViewAction("Install Realsense, Kinect, Orbbec and
Serial Card drivers", -1, AddressOf InstallDrivers, AddressOf
VerifyDrivers),
            New ListViewAction("Set default apps", 0, AddressOf
DefaultApps, AddressOf VerifyDefaultApps),

            New ListViewAction("Disable Window's abiity to turn off
devices", 0, AddressOf DisableTurnOff, AddressOf VerifyDisableTurnOff),

            New ListViewAction("Install Logitech Unifying Software", 0,
AddressOf InstallLogitech, AddressOf VerifyLogitech),

            New ListViewAction("Configure Notepad++", 0, AddressOf
ConfigureNotepad, AddressOf VerifyNotepad),
```

```
            New ListViewAction("Open file explorer to: This PC", 0,
AddressOf SetThisPC, AddressOf VerifyThisPC),

            New ListViewAction("Remove programs from desktop", 0,
AddressOf ClearDesktop, AddressOf VerifyDesktop),

            New ListViewAction("Create system restore point", 0,
AddressOf RestorePoint),

            New ListViewAction("Generate Sonar System Report", 0,
AddressOf SystemReport)

      }
End Class
```