DataTracker: A Comprehensive Artifact Management Tool for Embedded Chiller Applications

A Manuscript

Submitted to

the Department of Computer Science

and the Faculty of the

University of Wisconsin-La Crosse

La Crosse, Wisconsin

by

Graham T. Miller

in Partial Fulfillment of the

Requirements for the Degree of

Master of Software Engineering

April, 2025

DataTracker: A Comprehensive Artifact Management Tool for Embedded Chiller Applications

By Graham T. Miller

We recommend acceptance of this manuscript in partial fulfillment of this candidate's requirements for the degree of Master of Software Engineering in Computer Science. The candidate has completed the oral examination requirement of the capstone project for the degree.

Dr. Mao Zheng Examination Committee Chairperson	Date
Dr. Jason Sauppe Examination Committee Member	Date
Dr. Rig Das Examination Committee Member	Date

Abstract

Miller, Graham, T. "DataTracker: A Comprehensive Artifact Management Tool for Embedded Chiller Applications", Master of Software Engineering, April, 2025, Advisor: Zheng, Mao.

The Unit Controls team at Trane Technologies currently utilizes a hybrid desktop application that fetches data from a web server to manage all artifacts used to define user settings, informational data points, display text, and Low-Level Intelligence Device (LLID) I/O definitions. These artifacts are utilized by many embedded chiller control applications.

This manuscript describes the development of a replacement for an existing data management tool with many issues including high latency, lack of organizational ownership, as well as an outdated user interface with limited input validation. The replacement is designed to have improved latency, increased input validation, and better maintainability through a software development process that consistently included stakeholder feedback from the Unit Controls team at Trane Technologies. This manuscript highlights the overall design process, the challenges encountered during development, and the re-engineering approach taken to completely redesign and rebuild the *DataTracker* system.

Acknowledgements

I would first like to acknowledge my fiancé Rachel, for being incredibly supportive in my pursuit of higher education. Thank you for the countless sacrifices that you have made allowing me to achieve my goal. I could not have done this without you, thank you.

I would also like to thank my capstone advisor Dr. Mao Zheng for the many meetings providing incredibly useful feedback and direction over the course of this project, as well as her efforts as my professor for many of my courses in the MSE program.

Additionally, I would like to thank all the UWL faculty that have invested their time and effort in my success, as well as the Unit Controls team members at Trane Technologies, whose input made this project successful. Austin, Micah and Jason, your efforts were greatly appreciated. To my friends and family thank you for your incredible support and understanding over the last 3 years.

Thank you all for the time and energy that you have invested not only into the success of this project, but my development as a software engineer and a person.

Table of Contents

List of Figures

List of Tables

Glossary

Grey Box Testing

A method of testing software in which the person testing the application has knowledge of the internal implementation of the application, while testing it from the user perspective.

Pugh Decision Matrix

An analysis tool used to compare potential design choices against a set of selected criteria, allowing for a way to rank potential solutions based on their score, relative to other design choices.

PyQt

A Python wrapper for the Qt desktop application development framework, enabling the creation of cross-platform applications with user interfaces that closely resemble that of the user's operating system. PyQt6 provides a comprehensive set of Python bindings for the latest version of the Qt toolkit.

Qt Framework

A desktop application development framework, written in C++, designed to create cross-platform desktop applications and their associated user interfaces.

Qt Designer

A drag-and-drop UI design tool that allows developers to visually design user interfaces for Qt applications. Qt Designer automatically generates .ui files, which can be seamlessly integrated into PyQt or Qt desktop applications.

Sprint Task

Constructed by project stakeholders during the sprint planning phase of a sprint, sprint tasks define the work that is required to implement and support a user story. Each sprint will have a number of user stories, and each user story will have several sprint tasks that define the actual work required to implement the functionality described by that user story.

Story Point

A unit of measure used to estimate the effort (time and complexity) required to implement a user story.

Use Case

A description of how users interact with a system, outlining the flow of user inputs and the system’s response to those inputs.

User Story

A brief informal description of a functionality of a software product, from the user perspective. This is used to capture user-based requirements in Agile software development methodologies.

Velocity

A measure of the work that can be completed in a specific time frame. The rate at which a development team using the Agile SDLC model completes user stories, using the number of completed story points as a quantifier.

1. Introduction

1.1 Background

Trane Technologies is a corporation consisting of 45,000 employees at the time of writing (2025). Trane Technologies provides climate solutions specializing in the design and production of heating, ventilation, and air conditioning (HVAC) systems utilized in transportation, residential and commercial buildings. They are known for providing energy efficient HVAC solutions to customers such as data centers, manufacturers, hospitals and college campuses. Chillers are products manufactured and sold by Trane Technologies to provide the cold or hot water necessary in comfort cooling, comfort heating, and manufacturing processes.

The Unit Controls team at Trane Technologies is responsible for designing, developing, and maintaining embedded chiller applications used to optimally operate these large pieces of equipment. Each of these embedded applications require artifacts used to define user settings, informational data points, diagnostic information, Low-Level Intelligence Device (LLID) I/O, and display text strings translated into 27 supported languages.

Today these software artifacts are generated using *DataTracker,* a hybrid desktop application used to manage an Oracle database, served from Minneapolis, MN. *DataTracker's* primary users include the global Unit Controls development team, consisting of 30+ software developers, project technical leaders, and software test engineers.

Currently *DataTracker* manages many entities for each individual embedded controls application. Those entities include *Variables, Systems, Enumerations, Scalar Units*, *Abbreviations*, *Diagnostics*, *Devices*, and *Translations*. *Variables* include *Statuses* and *Settings*. *Statuses* are purely informational data points, such as a temperature reading. *Settings* are user settable options such as the temperature selection on a thermostat. *Systems* are utilized by variables and diagnostics describing where these entities are created within the embedded application. *Scalar Units* are utilized as potential datatypes for a variable. *Scalar Units* can consist of things like temperature or heating capacity. *Enumerations* are utilized as potential datatypes for a variable, containing a list of all allowable discrete values. For example, an operating mode enumeration may include things like Heating and Cooling modes. *Abbreviations* are used to generate unique identifiers associated with variables and diagnostics. *Diagnostics*, or alarms, are used to notify a user of a warning or critical issue pertaining to the chiller's operation. For example, a chiller's display may present the diagnostic "Comm Loss: Evaporator Water Temperature Sensor" to indicate that there has been a communication loss with the Evaporator Water Temperature Sensor. *Devices* pertain to the definition of inputs and outputs used by the Low-Level Intelligence Devices (LLIDs) on a chiller. This information is used to effectively facilitate communication between the chiller's main controller and the chiller's LLIDs. *Translations* include the text shown on any of the main controller's clients such as a touch display. *Translations* consist of a collection of text strings translated into 27 different supported languages. *Variables, Diagnostics, Devices, Systems, Scalar Units* and *Enumerations* all utilize one or several translations. These translations are provided to the main controller's clients to accurately display information in the selected language and proper format.

DataTracker also supports the export of each of the listed entities' information into formatted XML and binary artifacts that are consumed by the embedded software application. The information in many of these artifacts can also be exported into XLS format, an older binary-based Microsoft Excel file format, so that the information is easily readable and understandable by humans.

1.2 Need for Re-engineering

DataTracker was originally deployed in 2006. This application is a .Net web client written in C#, with a WinForms user interface, composed of over 120,000 lines of code. The original engineers that developed this application are no longer working for Trane Technologies and did not produce any accompanying documentation. This has proven to be an issue for maintenance and has left the Unit Controls team feeling uncertain about *DataTracker's* future. This lack of ownership has also made it difficult to facilitate improvements such as addressing known latency issues, improving input validation, and adding new desired functionality.

The author worked with *DataTracker's* stakeholders to assess the difficulty of refactoring the existing application to meet the needs of the application's users. Through much discussion it was decided that it would be more desirable to develop a new version of *DataTracker* using more modern technologies, rather than simply refactor the existing 19-year-old application.

The overall goal of this project was to design and develop a new system that addressed many of the issues faced by users of the original *DataTracker*. This included improving latency, maintainability, input validation, and adding additional functionality to increase the overall user experience.

2. Software Development Lifecycle Models

2.1 Models Considered

Several Software Development Life Cycle Models were evaluated during the inception of this project: Waterfall, Iterative, and Agile. The Waterfall model was quickly eliminated due to the expected churn or change in requirements from the effort of re-engineering and potential improvements. A large amount of effort in this project would be consumed in investigating the original implementation of *DataTracker* to fully capture its functionality in requirements. This means that as the project progressed, requirements could change according to these findings and as potential improvements were identified. This made the Agile and Iterative models front runners in this evaluation. The Agile and Iterative models are very similar, however differ in a few key areas. Agile incorporates stakeholder feedback continuously throughout the development process whereas the iterative approach gathers feedback at the end of each iteration. Agile also utilizes shorter development cycles called sprints. This decreased cycle time between iterations, or sprints, allows for Agile to be more collaborative and assists in the prioritization of continuously evolving requirements. Due to these key differences, the author selected the Agile approach for this project.

2.2 Model Used – Agile

The Agile Software Development Life Cycle Model is a cyclic model that breaks down a software development project into cycles called “sprints”. Each sprint will consist of the same main phases: sprint planning, requirement analysis, design, development or implementation, testing, and sprint retrospective. The next sprint in the project will begin immediately after the previous sprint has ended until all requirements have been satisfied, and the product is accepted by the customer. Figure 1 illustrates the Agile Software Development Life Cycle model.

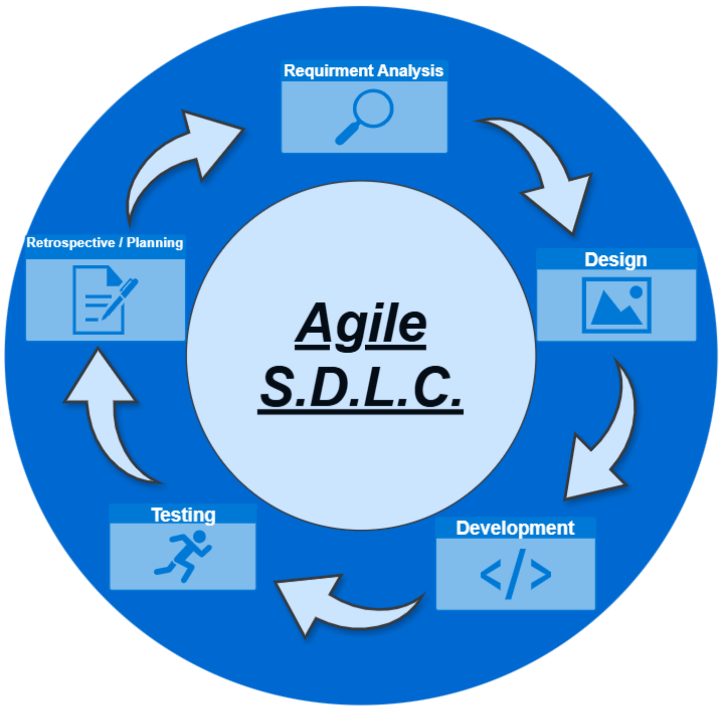
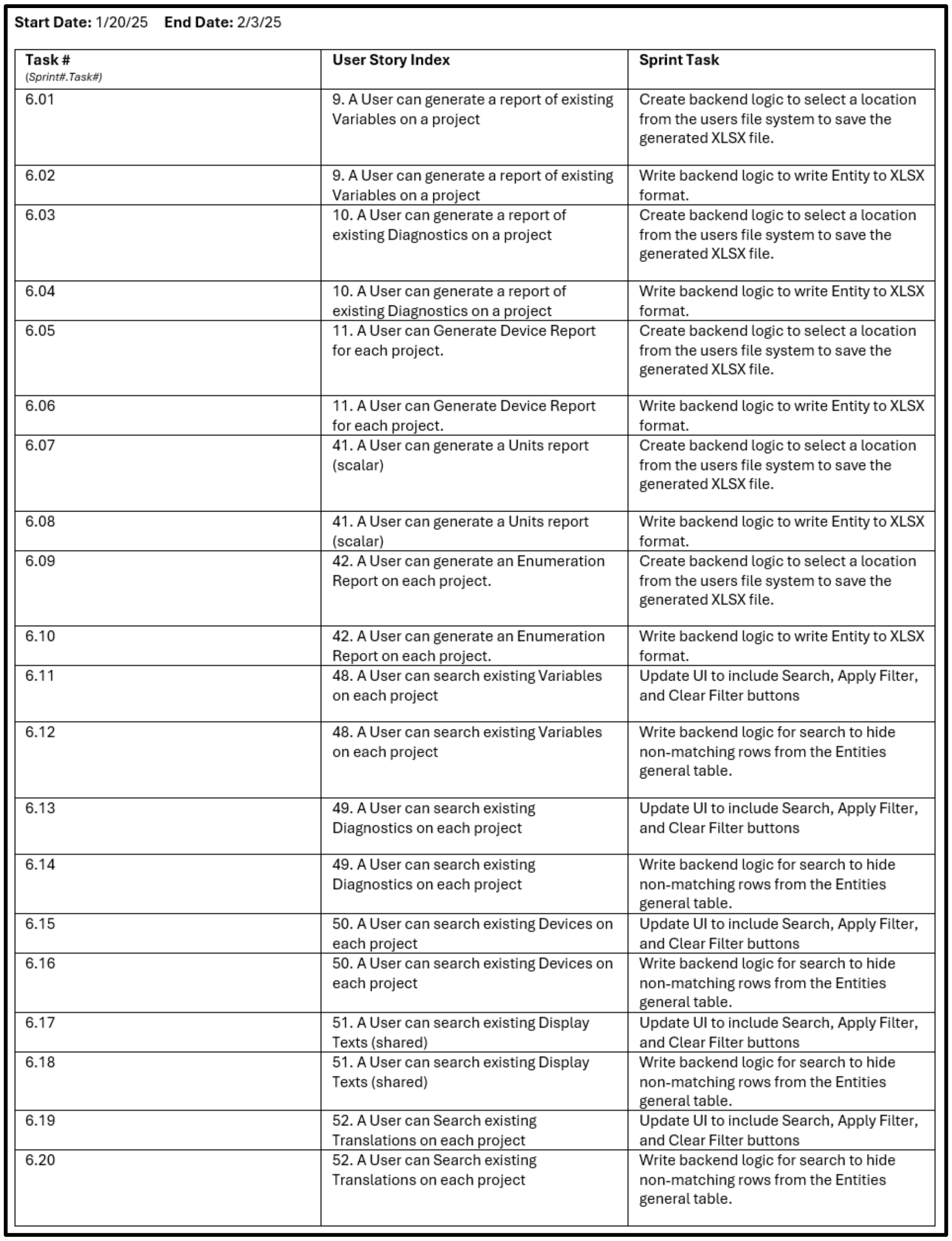


Figure 1: Agile SDLC – Sprint

During Sprint planning the user stories selected from the backlog were discussed and broken down into smaller implementable tasks, called sprint tasks. The author would document each of these sprint tasks in a sprint planning document to ensure that small actionable pieces of each user story were defined before starting development. This gave more resolution and granularity to the stakeholders about how much work was involved with each user story and helped to facilitate more accurate story point estimates. Table 1 shows an example of a sprint plan from sprint 6 of this project. Each sprint had an accompanying sprint task document.



Start Date: 1/20/25 **End Date:** 2/3/25

Task # *(Sprint#.Task#)*	**User Story Index**	**Sprint Task**
6.01	9. A User can generate a report of existing Variables on a project	Create backend logic to select a location from the users file system to save the generated XLSX file.
6.02	9. A User can generate a report of existing Variables on a project	Write backend logic to write Entity to XLSX format.
6.03	10. A User can generate a report of existing Diagnostics on a project	Create backend logic to select a location from the users file system to save the generated XLSX file.
6.04	10. A User can generate a report of existing Diagnostics on a project	Write backend logic to write Entity to XLSX format.
6.05	11. A User can Generate Device Report for each project.	Create backend logic to select a location from the users file system to save the generated XLSX file.
6.06	11. A User can Generate Device Report for each project.	Write backend logic to write Entity to XLSX format.
6.07	41. A User can generate a Units report (scalar)	Create backend logic to select a location from the users file system to save the generated XLSX file.
6.08	41. A User can generate a Units report (scalar)	Write backend logic to write Entity to XLSX format.
6.09	42. A User can generate an Enumeration Report on each project.	Create backend logic to select a location from the users file system to save the generated XLSX file.
6.10	42. A User can generate an Enumeration Report on each project.	Write backend logic to write Entity to XLSX format.
6.11	48. A User can search existing Variables on each project	Update UI to include Search, Apply Filter, and Clear Filter buttons
6.12	48. A User can search existing Variables on each project	Write backend logic for search to hide non-matching rows from the Entities general table.
6.13	49. A User can search existing Diagnostics on each project	Update UI to include Search, Apply Filter, and Clear Filter buttons
6.14	49. A User can search existing Diagnostics on each project	Write backend logic for search to hide non-matching rows from the Entities general table.
6.15	50. A User can search existing Devices on each project	Update UI to include Search, Apply Filter, and Clear Filter buttons
6.16	50. A User can search existing Devices on each project	Write backend logic for search to hide non-matching rows from the Entities general table.
6.17	51. A User can search existing Display Texts (shared)	Update UI to include Search, Apply Filter, and Clear Filter buttons
6.18	51. A User can search existing Display Texts (shared)	Write backend logic for search to hide non-matching rows from the Entities general table.
6.19	52. A User can Search existing Translations on each project	Update UI to include Search, Apply Filter, and Clear Filter buttons
6.20	52. A User can Search existing Translations on each project	Write backend logic for search to hide non-matching rows from the Entities general table.

Table 1: Sprint 6 Sprint Tasks

Several advantages to the Agile model include minimal upfront planning, rapid development, and consistent collaboration with the application’s stakeholders. All three of these advantages were observed over the course of this project. Due to the nature of the existing *DataTracker* application, minimal upfront planning was all that was possible without any clear owner of the products implementation. The project’s timeline was also aggressive given the overall scope of the project and the need to complete the project in about 6 months of development time. By breaking down the project into small more manageable sprints, the Agile model enabled the rapid development of the most important functionality to the customer in the time that the project allowed. One of the main goals of this project was to ensure that the new re-engineered *DataTracker* application did not rely on a single developer’s ownership or understanding. The Agile model lent itself well to this goal as each sprint facilitated consistent communication with the main stakeholders of the application. This allowed members of the Unit Controls team to be involved in the design of the overall product, leading to a more widespread understanding of the application. This consistent collaboration with stakeholders through sprint planning and requirement analysis also allowed for the project to adapt well to changes in requirements by allowing the team to prioritize remaining or additional work based on customer need.

Another benefit to using the Agile model for this project was the use of story points. Story points allowed for the team to quantify the effort required to implement a user story. This ability to quantify the work remaining on the project, or the work assigned in a sprint, allowed for a better understanding of the project’s overall progress. The author illustrates the ability to track project velocity using story points and burndown charts below in figures 2 and 3. This is a level of resolution that is not facilitated by other SDLC models such as Waterfall, where the only measure of velocity is the number of requirements implemented. This is important because requirements can differ greatly in the effort needed to implement them.

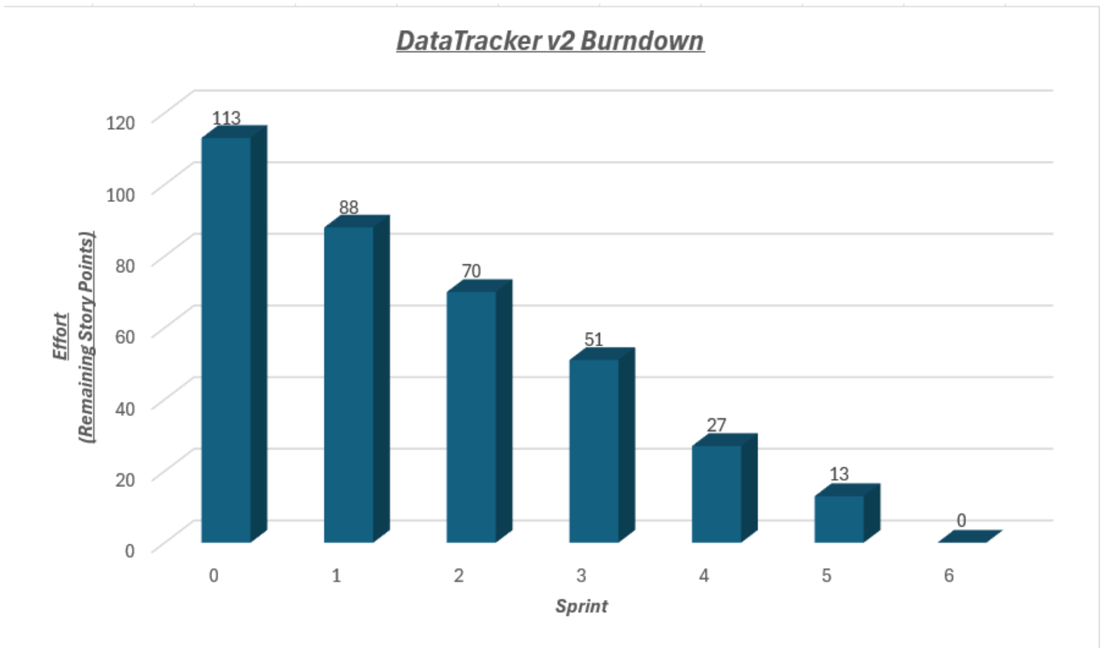


Figure 2: *DataTracker* v2 Burndown Chart

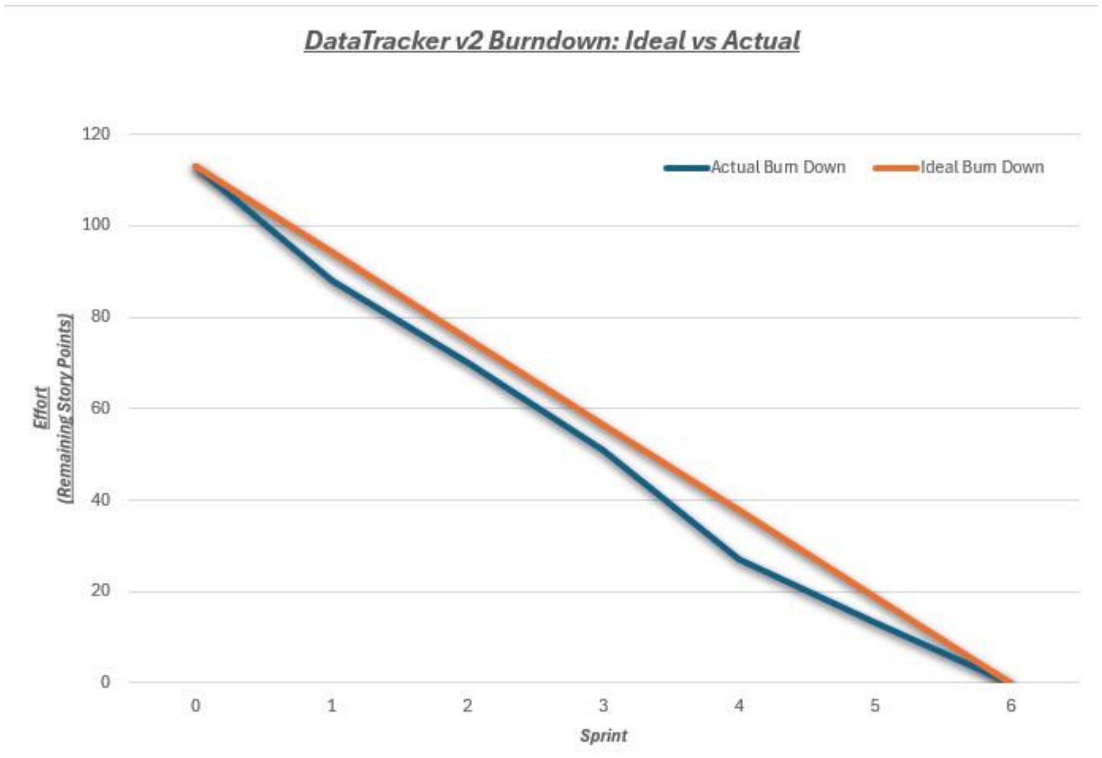


Figure 3: Ideal vs Actual Burndown Chart

In figures 2 and 3, these burn down charts show the actual and ideal number of remaining story points at the end of each sprint, and the starting total value of 113 before sprint 1 was completed, 0 on the x-axis. What is not in these illustrations is the number of user stories completed in each sprint.

This project had a total of 42 defined user stories, completed over 6 sprints. On average, 7 user stories were implemented and tested within each sprint. The number of user stories completed in each sprint could vary significantly based on the complexity involved in implementing each user story.

For example, in sprint 6, 9 user stories were implemented and tested. Each user story in sprint 6 had a relatively low story point value. Each user story within this sprint was evaluated as a 1 or a 2 by the team, meaning that the amount of time and complexity involved in implementing and testing these user stories was relatively low when compared to those completed in other sprints.

In short, not all user stories will require the same amount of time and complexity to implement and test. Evaluating the backlog of user stories and assigning story points allows for the team to divide the backlog into sprints of similar size and execution time. Over the course of this project, an average of 18.8333 story points were completed each sprint.

3. Requirements and Assumptions

3.1 Functional Requirements

Almost all the functional requirements for this system were derived from the original *DataTracker* application. The author worked with stakeholders to identify the core functionality of the original application as well as potential improvements, to mitigate any known shortcomings of the existing tool. Throughout the course of this project requirements were added, modified, and removed based on sprint retrospective, sprint planning and requirement analysis meetings with the project's stakeholders. This allowed for demonstration of the system under development's capabilities, and for the iterative refinement of requirements from the stakeholder perspective.

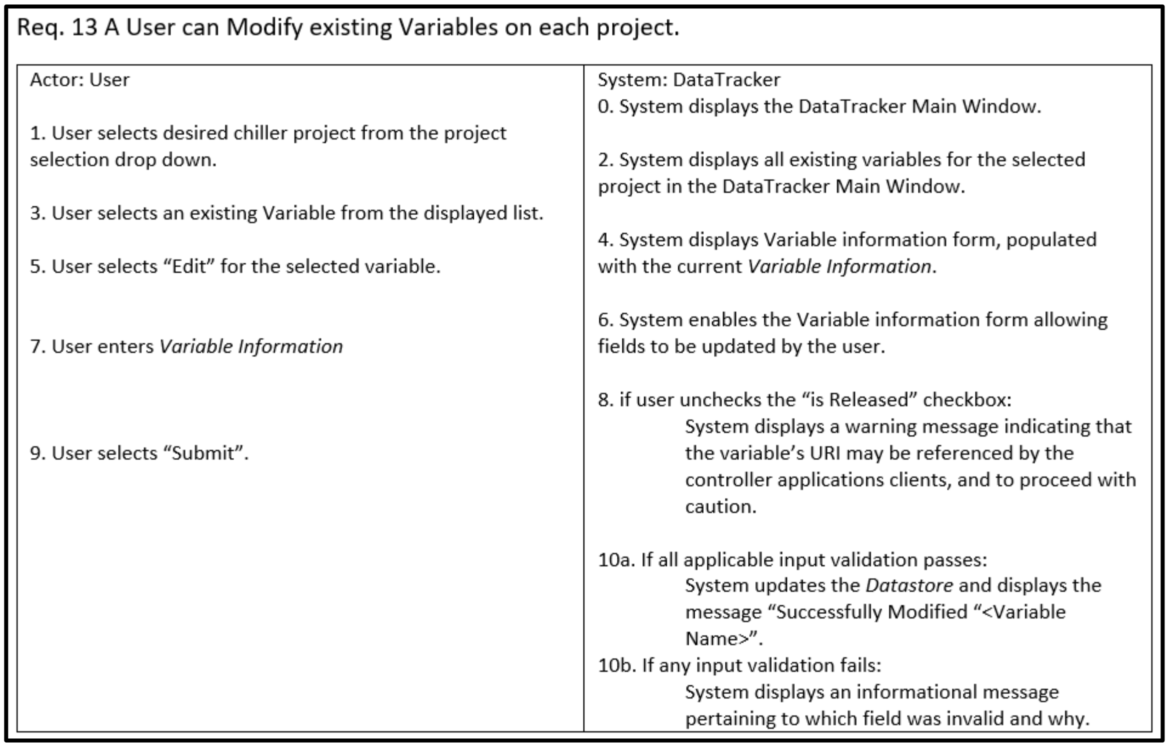
All functional requirements were defined as user stories with accompanying use cases in the project's requirements specification [3]. Listed below are all functional requirements identified.

- A User can View Variables on each project.
- A User can View Diagnostics on each project.
- A User can View Devices on each project.
- A User can View Translations on each project.
- A User can View Scalar Units (*Shared among projects)*.
- A User can View Enumerations on each project.
- A User can View Systems on each project.
- A User can View Abbreviations (*Shared among projects)*.
- A User can Add Variables on each project.
- A User can Modify existing Variables on each project.
- A User can Delete existing Variables on each project.

- A User can Add Diagnostics on each project.
- A User can Modify Diagnostics on each project.
- A User can Delete Diagnostics on each project.
- A User can Add Devices on each project.
- A User can Modify Devices on each project.
- A User can Delete Devices on each project.
- A User can Add Translations on each project.
    - Translation will be added to Global Translations file (S*hared among projects*).
    - Translation will be added to Message group (*Project specific)*.
- A User can Modify Translations on each project.
    - Translation will be Updated in Global Translations file (S*hared among projects*).
- A User can Delete Translations on each project.
- A User can Add Enumerations on each project.
- A User can Modify Enumerations on each project.
- A User can Delete Enumerations on each project.
- A User can Add Systems on each project.
- A User can Modify Systems on each project.
- A User can Delete Systems on each project.
- A User can Add Abbreviations (*Shared among projects)*.
- A User can Delete Abbreviations (*Shared among projects*).
- A User can view Display Texts (*Shared among projects*).

- A User can Add new Display Texts (Shared among projects).
- A User can modify Display Texts (*Shared among projects*).
- A User can generate a Scalar Units Report (*Shared among projects*).
- A User can generate an Enumeration Report for each project.
- A User can Generate a Variable Report for each project.
- A User can Generate a Diagnostic Report for each project.
- A User can Generate a Device Report for each project.
- A User can copy an existing Translation to a project.
- A User can view Shared Translations (*Shared among projects*).
- A User can search existing Variables.
- A User can search existing Diagnostics.
- A user can search existing Devices.
- A User can search Display Texts.
- A User can search existing Translations.

Figure 4 below captures one of the use cases from the project's requirement specification [3]. Note that the term *Variable Information*, in figure 4, is a reference to an entry in the requirement specification's glossary defining all attributes of a variable entity, as well as important input validation.



Req. 13 A User can Modify existing Variables on each project.

Actor: User	System: DataTracker
	0. System displays the DataTracker Main Window.
1. User selects desired chiller project from the project selection drop down.	
	2. System displays all existing variables for the selected project in the DataTracker Main Window.
3. User selects an existing Variable from the displayed list.	
	4. System displays Variable information form, populated with the current *Variable Information*.
5. User selects “Edit” for the selected variable.	
	6. System enables the Variable information form allowing fields to be updated by the user.
7. User enters *Variable Information*	
	8. if user unchecks the “is Released” checkbox: System displays a warning message indicating that the variable’s URI may be referenced by the controller applications clients, and to proceed with caution.
9. User selects “Submit”.	
	10a. If all applicable input validation passes: System updates the *Datastore* and displays the message “Successfully Modified “<Variable Name>”.
	10b. If any input validation fails: System displays an informational message pertaining to which field was invalid and why.

Figure 4: Use Case for Requirement #13

Like the use case shown in figure 4, each use case will reference the requirement’s unique identifier, as well as a description of the requirement within its title. Every requirement for this project, written as user stories, has an accompanying use case similar to what is shown in figure 4. Each use case will highlight the main actors involved in accomplishing what is specified in the associated requirement. The two-column format shows the actor’s interaction with the *DataTracker* system. This format allows for the high-level documentation of the flow of interactions between the users of the system and the system itself to drive a user-focused style of defining requirements.

3.2 Non-Functional Requirements

The system additionally had a collection of non-functional requirements that needed to be specified. The existing *DataTracker* application is very slow and depending on where the application is accessed from geographically it can take multiple minutes to navigate pages and complete actions. This latency was due to the central database of the existing application being served from Minneapolis, MN, as well as several design decisions made in the implementation of the original *DataTracker* application. The new system would need to have improved latency regardless of where its users were accessing it from. Additionally, the existing *DataTracker* application was poorly documented and was managed by employees who are no longer with the company. The new system would need to be more maintainable than the previous tool in its design, ensuring that it could be managed by new and existing resources.

3.3 Assumptions

The project's requirements specification [3] additionally included the following assumptions:

- *DataTracker* will rely on Trane Technologies credentials for access. And it will only be available for installation on the Trane Technologies network.
- *DataTracker* will rely on the Version Control System (VCS) for the following:
    - VCS will provide change history for all artifacts.
    - VCS will provide artifact revisions.
    - VCS will provide correct level of artifact access (Read/Write) for each user.
- *DataTracker* will NOT have a dedicated database upon the initial release. Design decisions will be made to allow for the addition of a global resource for translations, should the need become apparent in the future.

- All clients running *DataTracker* will utilize a Windows operating system.

4. Design

4.1 Architecture Overview

The author identified that the non-functional requirement of improved latency would be most heavily impacted by the application's architecture. To address this requirement, the author led a series of meetings with stakeholders of the *DataTracker* application. The goal of these meetings was to make a unified decision on what architecture would be used in the design of this new application. To assist in the analysis and selection of an architecture the author illustrated 3 high-level potential solutions. This illustration aided the team in understanding the context of what was being discussed. The team discussed the benefits and downsides to these 3 potential architectures, as well as others. After several meetings the team came to a unanimous decision.

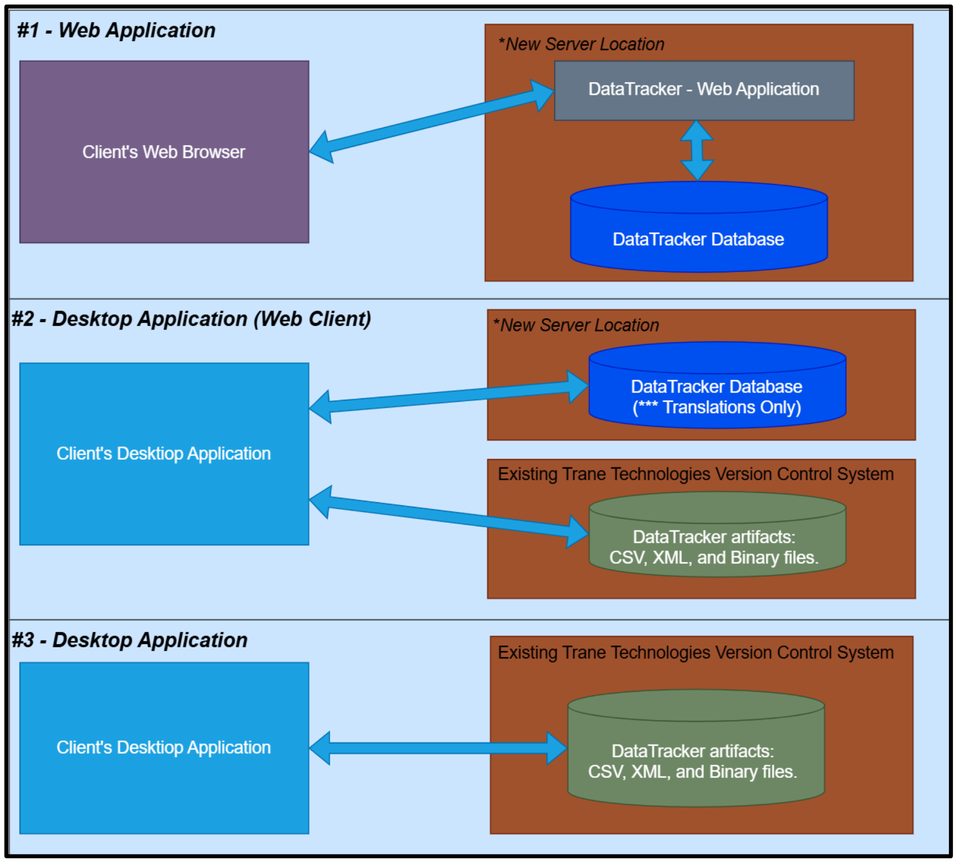
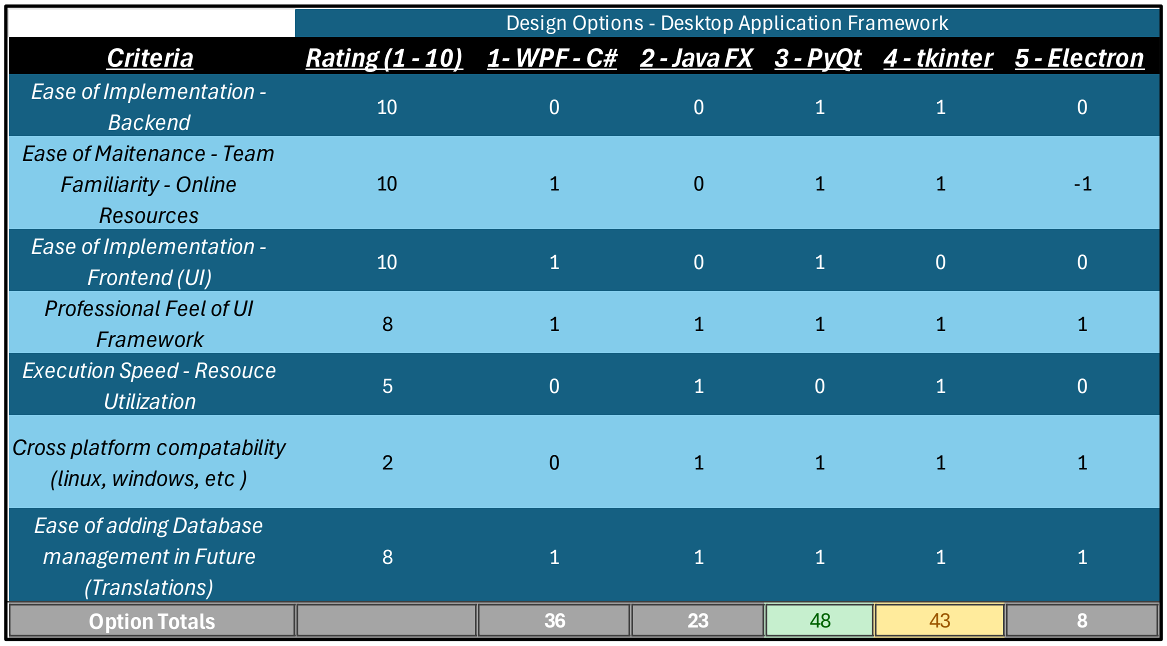

#1 - Web Application
Client's Web Browser
*New Server Location
DataTracker - Web Application
DataTracker Database
#2 - Desktop Application (Web Client)
Client's Desktiop Application
*New Server Location
DataTracker Database
(*** Translations Only)
Existing Trane Technologies Version Control System
DataTracker artifacts:
CSV, XML, and Binary files.
#3 - Desktop Application
Client's Desktiop Application
Existing Trane Technologies Version Control System
DataTracker artifacts:
CSV, XML, and Binary files.

Figure 5: Proposed Architecture Solutions

The project would move forward as a desktop application that leverages the existing version control system to store all embedded chiller application artifacts, option #3 in figure 5. However, this decision was made with one caveat: the application would be written in a way to support the potential separation of the translation entities into a database. The translation entities comprise most of the shared elements between all projects in *DataTracker's* datastore. This separation is shown in figure 5, proposed solution #2. This decision was based on the thought that if the number of users grew substantially, editing translation entities within the existing version control system could become a bottleneck for the users of the application. Although this was not a large concern within the Unit Controls organization, as these entities are changed infrequently enough to where this bottleneck should not take place, steps were taken in the design of the re-engineered *DataTracker* to ensure that any future changes in this area would have mitigated risk.

4.2 Technology Selection

To address the non-functional requirement of maintainability, the author set up several meetings to select the technologies that would be used to implement the new application. The author's intention was to select a technology that would not only satisfy the needs of the re-engineered application's functional requirements but ensure that the Unit Controls team felt comfortable maintaining and improving the application once it was implemented. After meeting with the team to identify potential technologies, the author created a Pugh Decision Matrix to rank the technologies based on the criteria discussed with the team.



		Design Options - Desktop Application Framework				
Criteria	***Rating (1 - 10)***	***1- WPF - C#***	***2 - Java FX***	***3 - PyQt***	***4 - tkinter***	***5 - Electron***
Ease of Implementation - Backend	10	0	0	1	1	0
Ease of Maitenance - Team Familiarity - Online Resources	10	1	0	1	1	-1
Ease of Implementation - Frontend (UI)	10	1	0	1	0	0
Professional Feel of UI Framework	8	1	1	1	1	1
Execution Speed - Resouce Utilization	5	0	1	0	1	0
Cross platform compatability (linux, windows, etc)	2	0	1	1	1	1
Ease of adding Database management in Future (Translations)	8	1	1	1	1	1
Option Totals		36	23	48	43	8

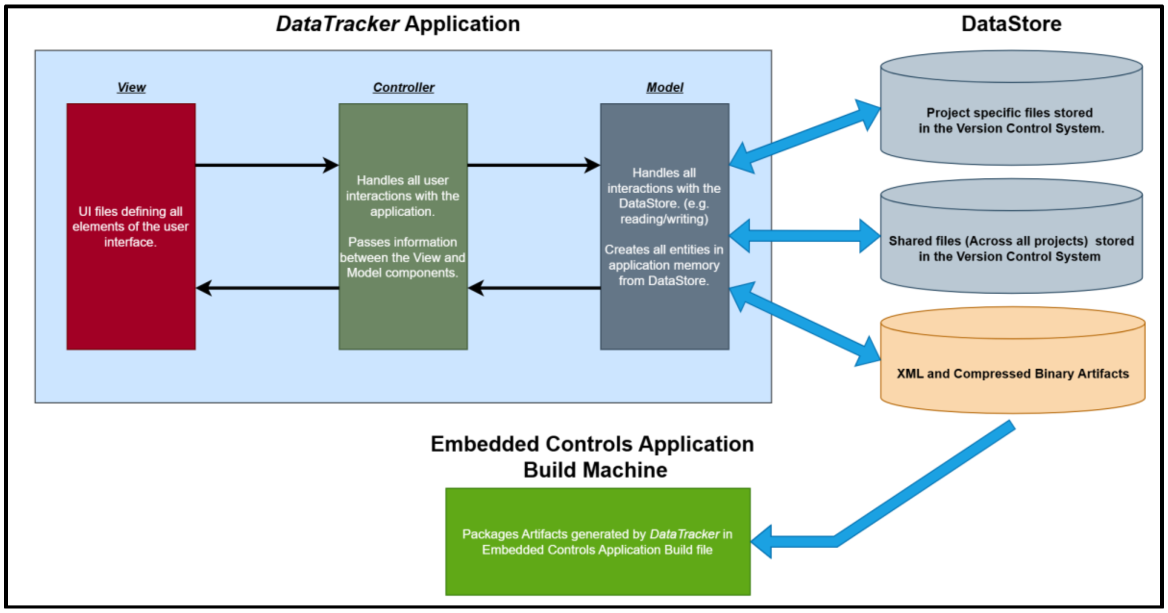
Figure 6: *DataTracker* - Pugh Decision Matrix – Technology Selection (Truncated)

This style of design analysis allowed for the team to weigh different criteria of each design choice separately based on each item's weight, or "rating". Through this exercise, the team determined that the PyQT framework would be leveraged in the implementation of this application. PyQt was chosen for many factors, but the most important being that the team felt comfortable supporting an application written in Python. Additionally, PyQT supports UI files generated by Qt Designer, a drag and drop UI design tool. This significantly reduced the amount of time and effort needed to develop the front end of the application.

4.3 MVC - Design Pattern

The author selected the Model View Controller (MVC) design pattern for the re-engineered *DataTracker* application. This was due to the fact that *DataTracker* would be a desktop application that heavily leveraged file I/O on the user's computer. This design pattern separates the application into 3 main components. The Model component handles all interactions between the application and the datastore. The Controller component will include all the functions necessary to manipulate entities in the Model and View components based on user interactions. The View component will contain all the entities that make up the user interface of the application.

As a result of the feedback from stakeholders in the architecture selection process, the author wanted to ensure that all data handling was encapsulated in one component of the application. This design decision would allow for the structure of the datastore to change in the future with minimized risk to the View and Controller components. This risk is further mitigated, as only the Model component would require refactoring if the need to incorporate a database for translation entities arises.



DataTracker Application
DataStore
View
Controller
Model
UI files defining all elements of the user interface.
Handles all user interactions with the application.
Passes information between the View and Model components.
Handles all interactions with the DataStore. (e.g. reading/writing)
Creates all entities in application memory from DataStore.
Project specific files stored in the Version Control System.
Shared files (Across all projects) stored in the Version Control System
XML and Compressed Binary Artifacts
Embedded Controls Application Build Machine
Packages Artifacts generated by *DataTracker* in Embedded Controls Application Build file

Figure 7: Application Overview – MVC Design Pattern

Figure 7 shows the *DataTracker* application and all three of its internal components, as well as the primary use case of the application and its datastore. Notice the datastore only interacts with the Model component of *DataTracker* and the Embedded Controls Application Build Machine.

4.4 Datastore Design

As described in the section 4.1 Architecture Overview, the datastore for the re-engineered *DataTracker* consists of files stored in a version control system. These files are divided based on their usage: files shared between all embedded chiller products are stored in one location, while product-specific files are stored alongside the source code for those applications. This pattern was primarily driven by the ease of access for the embedded chiller application build machine, as well as providing the ability to create stable versions of released software products.

Storing these files in the version control system allows for datastore revisions, ensuring that no additional or unwanted changes impact the artifacts generated for use in these embedded chiller applications. This has been a pain point for the global Unit Controls development team as multiple projects often run in parallel on the same embedded chiller application. With multiple projects modifying the same revision-less database, it is not uncommon for projects to inherit unwanted or non-applicable changes. These non-applicable changes necessitate additional validation, and a rebuild of the application before releasing the software, resulting in project delays and increased costs.

Almost all the files that are used to support the re-engineered *DataTracker* are stored in CSV format. The CSV format was chosen for its speed in file I/O operations over formats like XML. This format allows for all files utilized by this tool to be easily readable and understandable by humans, as well as allowing for the direct import of these files into a database should the need for a traditional database arise in the future.

The design of the datastore was significantly impacted by the existence of the original *DataTracker's* database, which contained almost 20 years' worth of data. The new datastore would need to support this existing data without impacting the generated artifacts used by the embedded chiller applications. This meant that many existing patterns within the original database had to be maintained in the new format. Changes were based on identifying unused or poorly formatted entities within the existing database, with the goal of simplifying the datastore to only contain the data needed to generate the artifacts consumed by the embedded chiller applications.

One example of this redesign relates to the *Diagnostic* entity shown in figure 8. In the original *DataTracker* database, this entity consisted of four separate tables and a collection of 38 total attributes across those tables. By identifying the required attributes for the re-engineered *DataTracker* application the author simplified this into one entity with eight attributes. Most of the attributes from the existing *DataTracker's* diagnostic entities were redundant or no longer used.

Throughout the project the author ensured that any effort to simplify and re-design these entities was evaluated by *DataTracker's* stakeholders at the beginning and end of each sprint. The outcome of this redesign is illustrated in figure 8, which is extracted from the *DataTracker* software design specification [4]. The new datastore includes only 16 separate entities, a significant reduction from the original 77 tables. Much of this simplification was a result of the original *DataTracker* having to support user access, user roles, entity locking, and additional functionalities to support 3 generations of embedded controller hardware. In the re-engineered *DataTracker* much of the user and project identification functionality could be eliminated, as it is provided by the existing version control system that stores this data. Additionally, only the current and future generations of controller hardware would be supported by the re-engineered *DataTracker*, decreasing the number of features that the application needs to support.

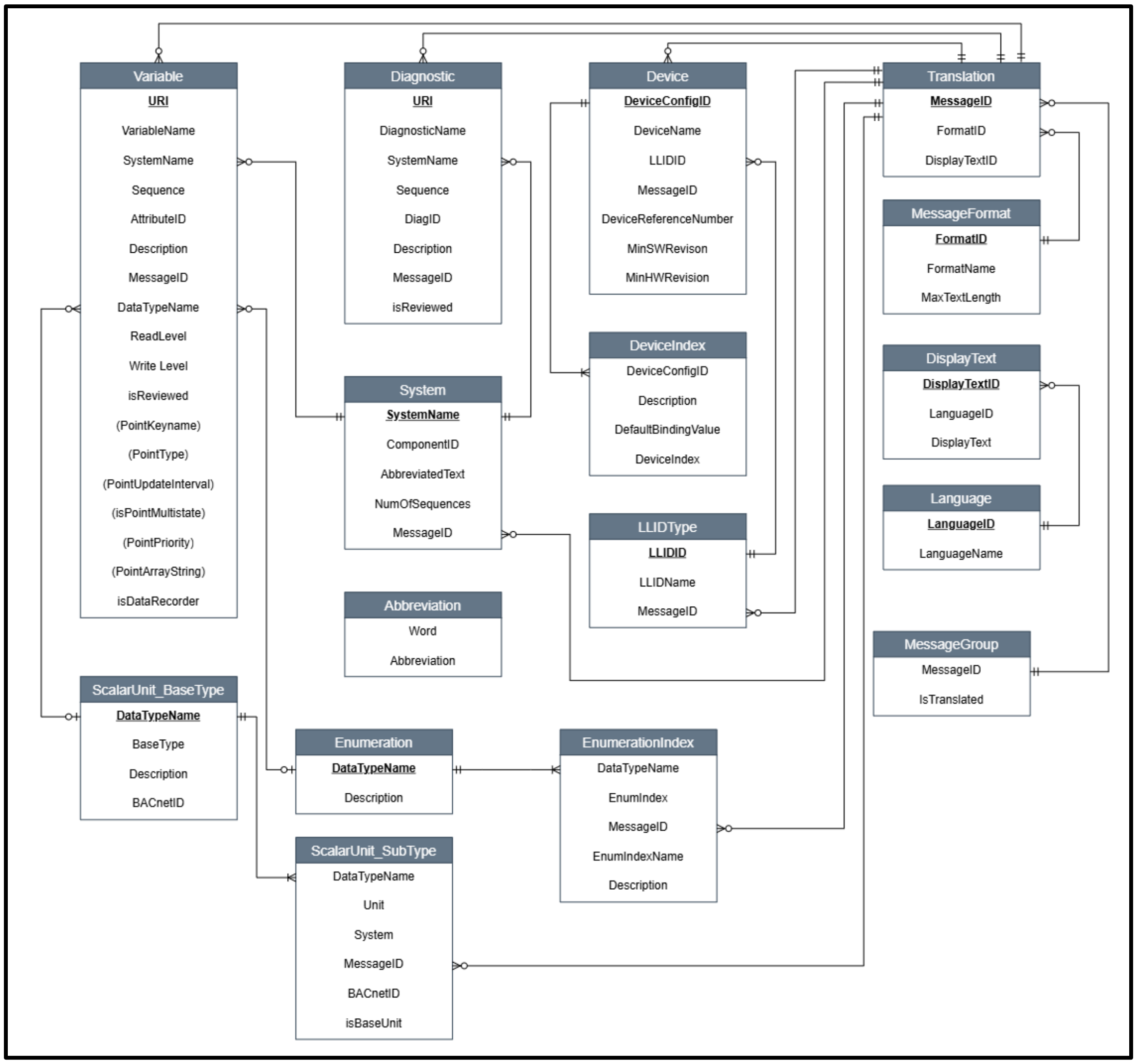


Figure 8: *DataTracker* - Datastore Entity Relationship Diagram

Much consideration was given to the relationship between the entities shown in figure 8. The four main entities illustrated: *Variables*, *Diagnostics*, *Devices*, and *Translations*, are positioned at the top of the diagram. All supporting entities are shown with their respective relationship to these main entities.

Most notably, the *Translation* entity is utilized by almost all entities in this datastore. This relationship is necessary to provide embedded chiller control application clients, such as a physical display and technician tool, with the proper text associated with each entity. *Variables*, *Diagnostics*, and *Devices* are entirely chiller product specific. *Translations* and their supporting entities, however, are shared between all chiller applications.

By abstracting the selected translated texts to a *Message ID* for all entities to reference, this design enables flexibility for future redesigns of this area of the datastore. Should the shared *Translation* files in the version control system become a bottleneck for *DataTracker*'s users, this data could be separated into another tool or datastore all together. Depending on how this separation is designed and implemented, product-specific entities could reference a *Message ID* in any global resource. This approach limits the amount of risk and necessary changes required for any re-design surrounding all other entities in the re-engineered datastore.

4.5 UI Design

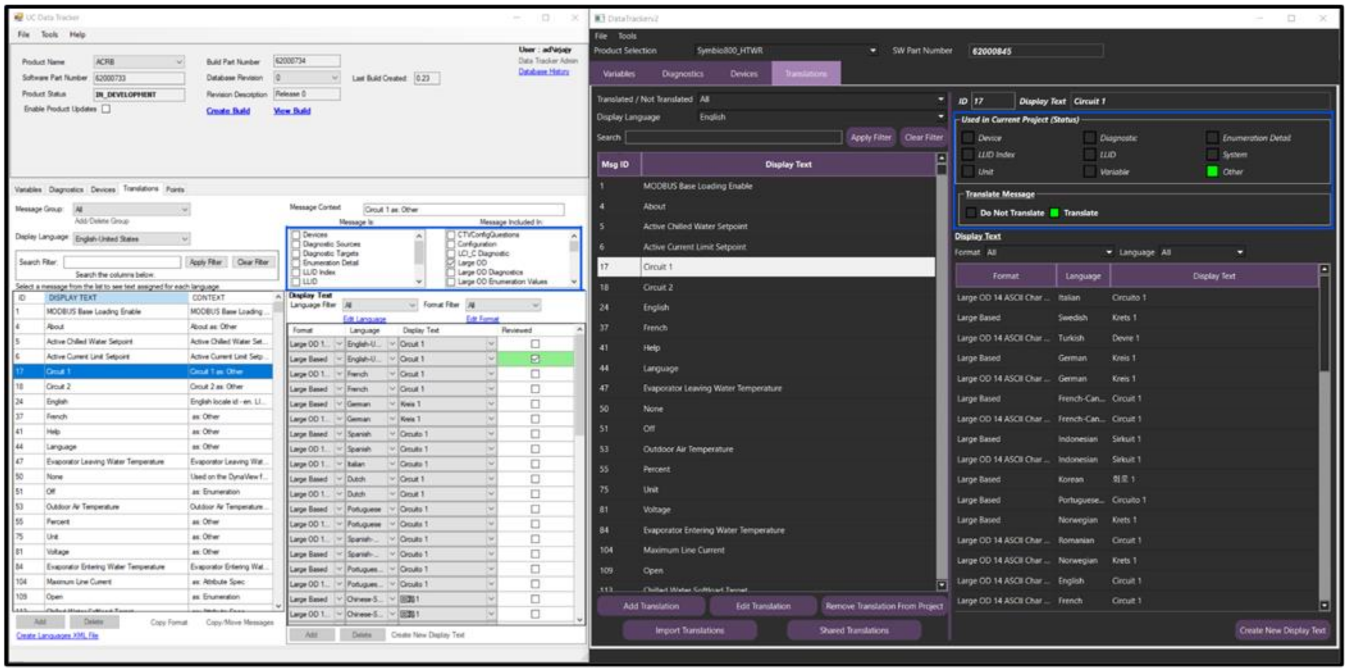
The re-engineered application's UI design was heavily influenced by the original *DataTracker* application. This was desirable as it would allow the users of *DataTracker* to leverage their familiarity with the existing application. This in theory would limit the amount of re-training necessary for users to work with the re-engineered *DataTracker* successfully.

This decision also enabled the rapid development of the frontend of the application. The author was able to baseline almost all pages of the UI within the first sprint. This allowed for the refinement of the design of the UI over the course of the project. This refinement was driven by stakeholders' feedback given in the sprint planning and sprint retrospective meetings held at the beginning and end of each sprint.

An example of this refinement is the UI design of the Translations page. One of the user pain points targeted in this re-design was the confusion generated by the format of the "Message is" and "Message Included In" group boxes shown in figure 9, bordered in blue.

The purpose of the “Message Included In” group box in the original *DataTracker* was to show what entities were using a Translation across all projects. Due to how this was formatted in the UI, the user would have to scroll through the entirety of the list to see all the entity contexts where a translation was used. Additionally, due to this group box resembling other group boxes in the application, users would often believe that they could edit these fields when they were purely informational. Through continuous feedback from *DataTracker*’s stakeholders over the course of this project, areas like this could easily be discussed and improved upon. In the re-engineered application, this area was re-designed as the “Used in Current Project (Status)” group box. In working with the project’s stakeholders, the author found that it would be more useful to show this information as project-specific, as that is the context that most users are working from. The layout of this area was also improved to show all entity contexts, to make this information more readable without user intervention (scrolling). The format of the text in this area was also changed to show that it was informational only and not editable. This was accomplished by greying and italicizing the text for each category and listing “(Status)” in the name of the group box.

The purpose of the “Message Included In” group box was to select translations for exporting to a translation service utilized by the Unit Controls team to provide translated text strings in 27 languages. This area in the original *DataTracker* had ***19*** categories leading to user confusion, and common misuse. The author and stakeholders agreed that this could be significantly simplified to 2 categories: Translate and Do Not Translate.


UC Data Tracker
File Tools Help
Create Build
View Build
User : ad\vijajy
Data Tracker Admin
Database History
Variables Diagnostics Devices Translations Points
Add/Delete Group
Apply Filter Clear Filter
Search the columns below.
Select a message from the list to see text assigned for each language.
Edit Language
Edit Format
Add Delete Copy Format Copy/Move Messages
Create Languages XML File
Add Delete Create New Display Text
DataTracker v2
File Tools
Variables Diagnostics Devices Translations
Apply Filter Clear Filter
Add Translation Edit Translation Remove Translation From Project
Import Translations Shared Translations
Create New Display Text

Product Name: ACRB
Software Part Number: 62000733
Product Status: IN_DEVELOPMENT
Enable Product Updates: ☐
Build Part Number: 62000734
Database Revision: 0
Revision Description: Release 0
Last Build Created: 0.23

Message Group: All
Display Language: English-United States
Search Filter:
Message Context: Circuit 1 as: Other

Message Is	Message Included In
☐ Devices	☐ CTVConfigQuestions
☐ Diagnostic Sources	☐ Configuration
☐ Diagnostic Targets	☐ LCI_C Diagnostic
☐ Enumeration Detail	☑ Large OD
☐ LLID Index	☐ Large OD Diagnostics
☐ LLID	☐ Large OD Enumeration Values

ID	DISPLAY TEXT	CONTEXT
1	MODBUS Base Loading Enable	MODBUS Base Loading ...
4	About	About as: Other
5	Active Chilled Water Setpoint	Active Chilled Water Set...
6	Active Current Limit Setpoint	Active Current Limit Setp...
17	Circuit 1	Circuit 1 as: Other
18	Circuit 2	Circuit 2 as: Other
24	English	English locale id - en. U...
37	French	as: Other
41	Help	as: Other
44	Language	as: Other
47	Evaporator Leaving Water Temperature	Evaporator Leaving Wat...
50	None	Used on the DynaView f...
51	Off	as: Enumeration
53	Outdoor Air Temperature	Outdoor Air Temperature...
55	Percent	as: Other
75	Unit	as: Other
81	Voltage	as: Other
84	Evaporator Entering Water Temperature	Evaporator Entering Wat...
104	Maximum Line Current	as: Attribute Spec
109	Open	as: Enumeration

Display Text
Language Filter: All Format Filter: All

Format	Language	Display Text	Reviewed
Large OD 1...	English-U...	Circuit 1	☐
Large Based	English-U...	Circuit 1	☑
Large OD 1...	French	Circuit 1	☐
Large Based	French	Circuit 1	☐
Large Based	German	Kreis 1	☐
Large OD 1...	German	Kreis 1	☐
Large Based	Spanish	Circuito 1	☐
Large OD 1...	Spanish	Circuito 1	☐
Large OD 1...	Italian	Circuito 1	☐
Large Based	Dutch	Circuit 1	☐
Large OD 1...	Dutch	Circuit 1	☐
Large Based	Portuguese	Circuito 1	☐
Large OD 1...	Portuguese	Circuito 1	☐
Large OD 1...	Spanish-...	Circuito 1	☐
Large Based	Spanish-...	Circuito 1	☐
Large Based	Portugues...	Circuito 1	☐
Large OD 1...	Portugues...	Circuito 1	☐
Large Based	Chinese-S...	[illegible]1	☐
Large OD 1...	Chinese-S...	[illegible]1	☐

Product Selection: Symbio800_HTWR
SW Part Number: 62000845
Translated / Not Translated: All
Display Language: English
Search:

Msg ID	Display Text
1	MODBUS Base Loading Enable
4	About
5	Active Chilled Water Setpoint
6	Active Current Limit Setpoint
17	Circuit 1
18	Circuit 2
24	English
37	French
41	Help
44	Language
47	Evaporator Leaving Water Temperature
50	None
51	Off
53	Outdoor Air Temperature
55	Percent
75	Unit
81	Voltage
84	Evaporator Entering Water Temperature
104	Maximum Line Current
109	Open

ID: 17 Display Text: Circuit 1

Used in Current Project (Status)

☐ Device	☐ Diagnostic	☐ Enumeration Detail
☐ LLID Index	☐ LLID	☐ System
☐ Unit	☐ Variable	🟩 Other

Translate Message
☐ Do Not Translate 🟩 Translate

Display Text
Format: All Language: All

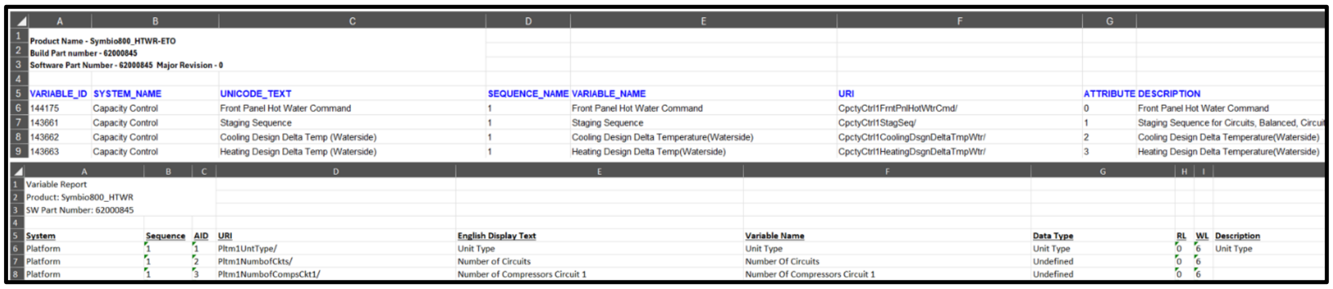
Format	Language	Display Text
Large OD 14 ASCII Char ...	Italian	Circuito 1
Large Based	Swedish	Krets 1
Large OD 14 ASCII Char ...	Turkish	Devre 1
Large Based	German	Kreis 1
Large OD 14 ASCII Char ...	German	Kreis 1
Large Based	French-Can...	Circuit 1
Large OD 14 ASCII Char ...	French-Can...	Circuit 1
Large Based	Indonesian	Sirkuit 1
Large OD 14 ASCII Char ...	Indonesian	Sirkuit 1
Large Based	Korean	회로 1
Large Based	Portuguese...	Circuito 1
Large Based	Norwegian	Krets 1
Large OD 14 ASCII Char ...	Romanian	Circuit 1
Large OD 14 ASCII Char ...	Norwegian	Krets 1
Large OD 14 ASCII Char ...	English	Circuit 1
Large OD 14 ASCII Char ...	French	Circuit 1

Figure 9: Project Translations Page - Before and After– Translation Context and Translation Is Translated

Although this is only one example of improvement, almost every page of *DataTracker's* UI was re-designed to some degree. Several pages were added and removed to avoid user confusion and misuse. This example is used to highlight the process used for UI design refinement. These design discussions would typically take place during the Sprint planning meeting before each sprint, including the author and stakeholders analyzing the pages of the original tool and identifying common user pain points. These changes would then be executed over the course of the associated sprint and demonstrated in the Sprint Retrospective meeting to ensure stakeholder acceptance of the UI design. The before and after for all of the application's pages can be seen in the appendix of this manuscript.

4.6 Reports

The re-engineered *DataTracker* application supports report generation for many of its entities into XLSX format, Microsoft Excel's current XML based file format that is easily readable by humans. Alike to the process followed in the UI re-design, reports were evaluated in Sprint Planning and Sprint Retrospective meetings. Figure 10 shows the before and after of the Variable Report from the same product. In this figure the columns differ in order, name and content. The columns were re-organized and modified to more closely reflect how these reports are utilized by *DataTracker's* stakeholders. Examples of this redesign include removing duplicate or unused columns, modifying column names to more accurately describe the column's content, and moving the most utilized information to the left-most position in the report.



Product Name - Symbio800_HTWR-ETO
Build Part number - 62000845
Software Part Number - 62000845 Major Revision - 0

VARIABLE_ID	SYSTEM_NAME	UNICODE_TEXT	SEQUENCE_NAME	VARIABLE_NAME	URI	ATTRIBUTE	DESCRIPTION
144175	Capacity Control	Front Panel Hot Water Command	1	Front Panel Hot Water Command	CpctyCtrl1FrntPnlHotWtrCmd/	0	Front Panel Hot Water Command
143661	Capacity Control	Staging Sequence	1	Staging Sequence	CpctyCtrl1StagSeq/	1	Staging Sequence for Circuits, Balanced, Circui
143662	Capacity Control	Cooling Design Delta Temp (Waterside)	1	Cooling Design Delta Temperature(Waterside)	CpctyCtrl1CoolingDsgnDeltaTmpWtr/	2	Cooling Design Delta Temperature(Waterside)
143663	Capacity Control	Heating Design Delta Temp (Waterside)	1	Heating Design Delta Temp(Waterside)	CpctyCtrl1HeatingDsgnDeltaTmpWtr/	3	Heating Design Delta Temperature(Waterside)

Variable Report
Product: Symbio800_HTWR
SW Part Number: 62000845

System	Sequence	AID	URI	English Display Text	Variable Name	Data Type	RL	WL	Description
Platform	1	1	Pltm1UntType/	Unit Type	Unit Type	Unit Type	0	6	Unit Type
Platform	1	2	Pltm1NumbofCkts/	Number of Circuits	Number Of Circuits	Undefined	0	6	
Platform	1	3	Pltm1NumbofCompsCkt1/	Number of Compressors Circuit 1	Number Of Compressors Circuit 1	Undefined	0	6	

Figure 10: Variable Report Before and After

4.7 Security

The re-engineered *DataTracker's* system security will be provided through the Trane Technologies network and the existing version control system. *DataTracker* is a data-centric application, and from the security perspective the datastore and application source code is what needs protection from potential attackers.

To gain access to the datastore an employee at Trane Technologies must first request access to the network location in which the version control system lives. That employee must also request access to the specific repository in which this data is stored through a controlled process with multiple levels of verification that requires manager approval. The *DataTracker* application's installer will also live on the version control system and will require a similar degree of access to install and to view source code.

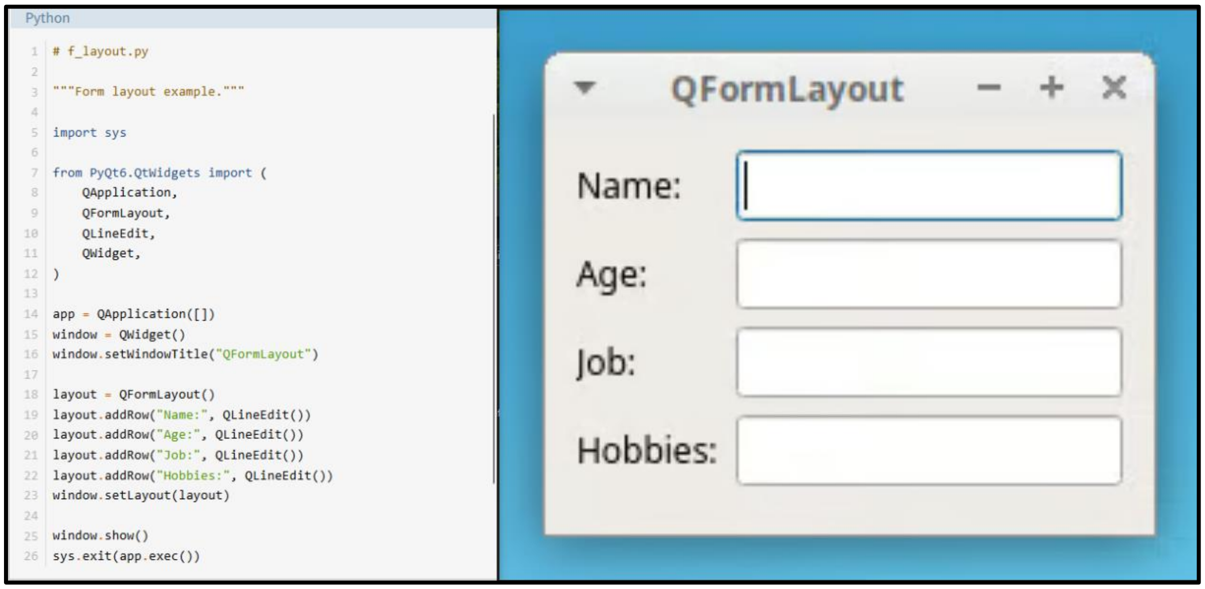
The version control system also supports various levels of access for groups of users. This would allow employees that are not Unit Controls software engineers to view the datastore and install the *DataTracker* application but not modify the data or the application source code. Additionally, for artifacts that are shared among all applications, the group of users that are allowed to modify these entities could be limited to only specific individuals if necessary.

This level of security is acceptable for this application and most small in-house applications at Trane Technologies, as this application and its datastore will only be distributed to Trane Technologies personnel.

5. Implementation

As mentioned in section 4.2 Technology Selection, the PyQT framework was selected for the implementation of this application. PyQt is a Python wrapper for the widely used Qt C++ framework. The Qt framework is known for creating applications that run on multiple operating systems and is used across many different industries. Qt is most notably used to build out Tableau's user interface, as well as the UX for automotive applications such as Spotify's "car thing" [1].

A simple example of the style of programming associated with PyQt can be seen in figure 11, captured in a tutorial from the Python documentation website, Real Python [2]. In this example we can see the Python code used to generate the simple example UI of QFormLayout.



```python
# f_layout.py

"""Form layout example."""

import sys

from PyQt6.QtWidgets import (
    QApplication,
    QFormLayout,
    QLineEdit,
    QWidget,
)

app = QApplication([])
window = QWidget()
window.setWindowTitle("QFormLayout")

layout = QFormLayout()
layout.addRow("Name:", QLineEdit())
layout.addRow("Age:", QLineEdit())
layout.addRow("Job:", QLineEdit())
layout.addRow("Hobbies:", QLineEdit())
window.setLayout(layout)

window.show()
sys.exit(app.exec())
```

Figure 11: Simple PyQT Example - Real Python

PyQt also supports the use of a drag and drop UI design tool, called Qt Designer. The Qt Designer tool was leveraged in the design and implementation of the re-engineered *DataTracker's* frontend. In figure 12, the main window for the "Translated Text" page of *DataTracker* is shown. Within figure 12, the QPlainTextEdit *messageID* is selected. The files generated by Qt designer are saved with the ui file extension and can be loaded directly into a PyQt application.

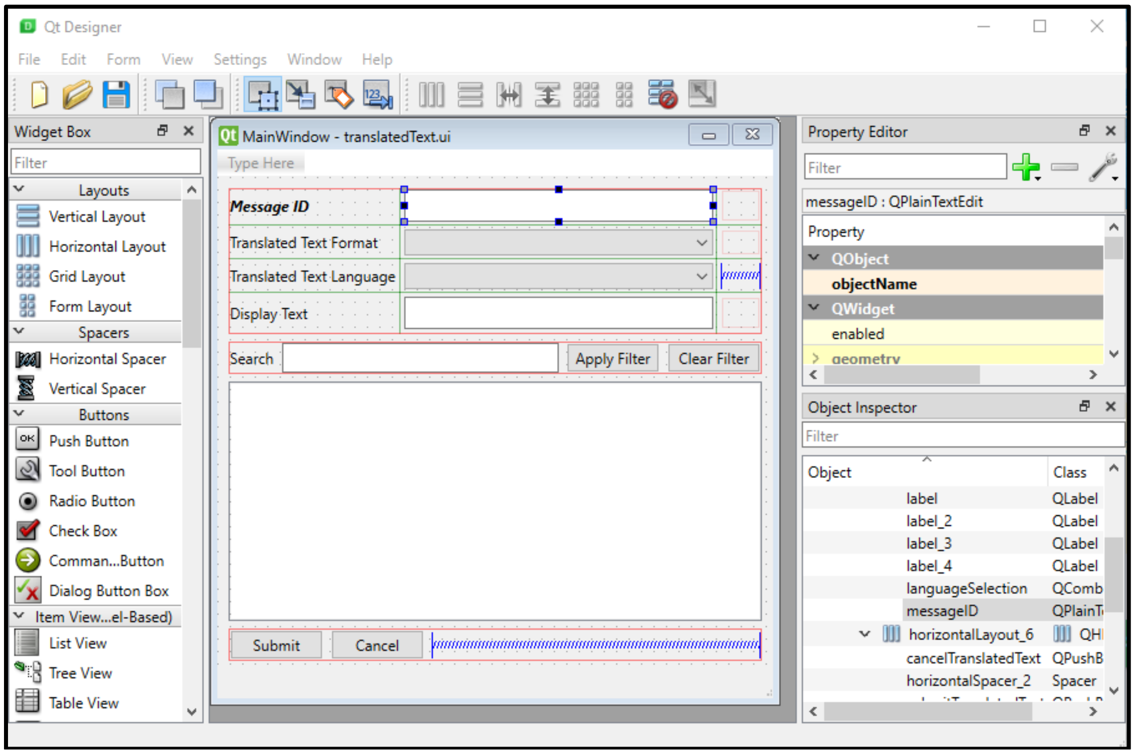
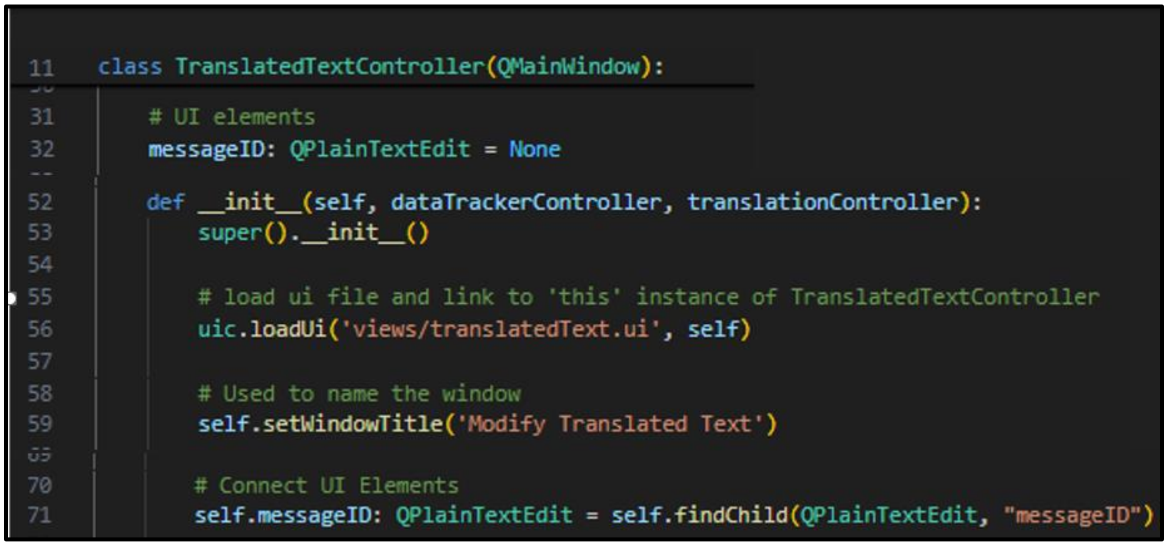


Figure 12: Qt Designer - Translated Texts Page

Within the re-engineered DataTracker application the QPlainTextEdit, shown in figure 12 can be directly referenced within the application by its name *messageID*. This loading of the *translatedText* ui file can be seen in figure 13. After the file has been loaded and associated with an instance of the *TranslatedTextController* class, the PyQt application can then find any specific element in that ui file using the *QMainWindow*::*findChild( )* operation. In figure 13 this is shown by instantiating the class variable *messageID* with the QPlainTextEdit child element of the translatedText ui file, *messageID*.



```
class TranslatedTextController(QMainWindow):

    # UI elements
    messageID: QPlainTextEdit = None

    def __init__(self, dataTrackerController, translationController):
        super().__init__()

        # load ui file and link to 'this' instance of TranslatedTextController
        uic.loadUi('views/translatedText.ui', self)

        # Used to name the window
        self.setWindowTitle('Modify Translated Text')

        # Connect UI Elements
        self.messageID: QPlainTextEdit = self.findChild(QPlainTextEdit, "messageID")
```

Figure 13: Loading ui elements from translatedText.ui

Utilizing the PyQt framework with Qt Designer lent itself well to the Model View Controller (MVC) design pattern as the UI files generated by Qt Designer encapsulated the View component of the application. The controllers with in the re-engineered *DataTracker* would load the associated UI files and store the necessary logic to populate the UI elements with data, as well as handling user interactions with the View component. The Controllers would then load and save modified data through the Model component of the application, which handles all the application’s interactions with the datastore.

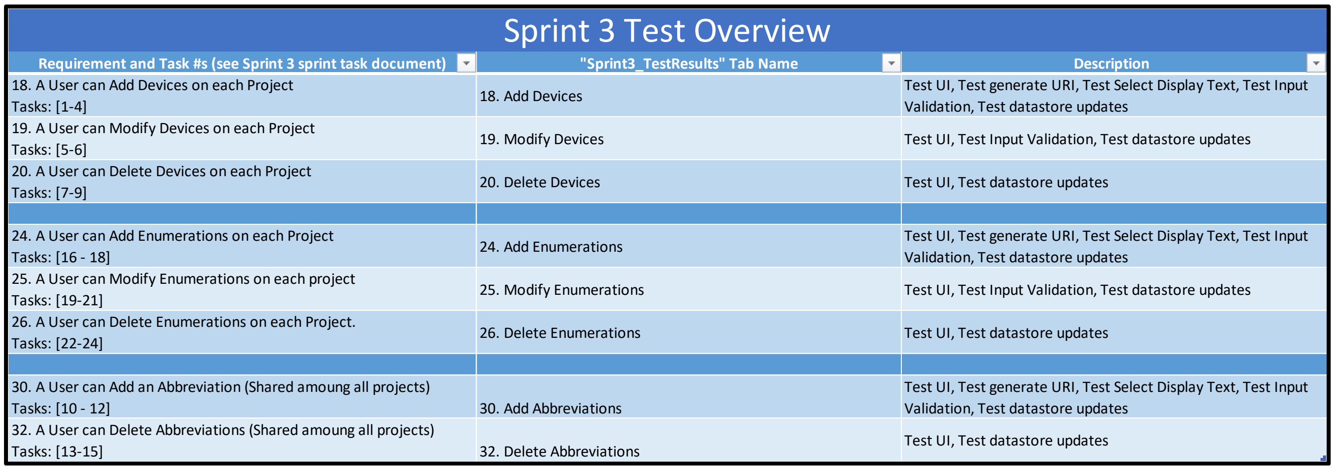
6. Testing

As is typical to the Agile approach, testing took place as part of each individual sprint. The author performed all testing throughout this project, though demonstrations of the tool's functionality to *DataTracker's* stakeholders occurred at the beginning and end of each sprint. The author utilized a grey box testing style, as the implementation was known to the author, but most tests were created from the user perspective. Each user story was tested individually as test cases were generated for each sprint.

The test cases generated in each sprint were all formatted into one document for each sprint. Each sprint testing document included an overview as well as individual tabs for each user story tested in that sprint.

Table 2 shows the sprint 3 test document's overview tab. In every sprint testing document, the overview tab was formatted to include the requirement and task number, tab name, and description columns with a row for each user story associated with the sprint. The requirement and task number column will reference the user story index, the user story, and the sprint tasks associated with that user story from the sprint task document created at the beginning of each sprint.

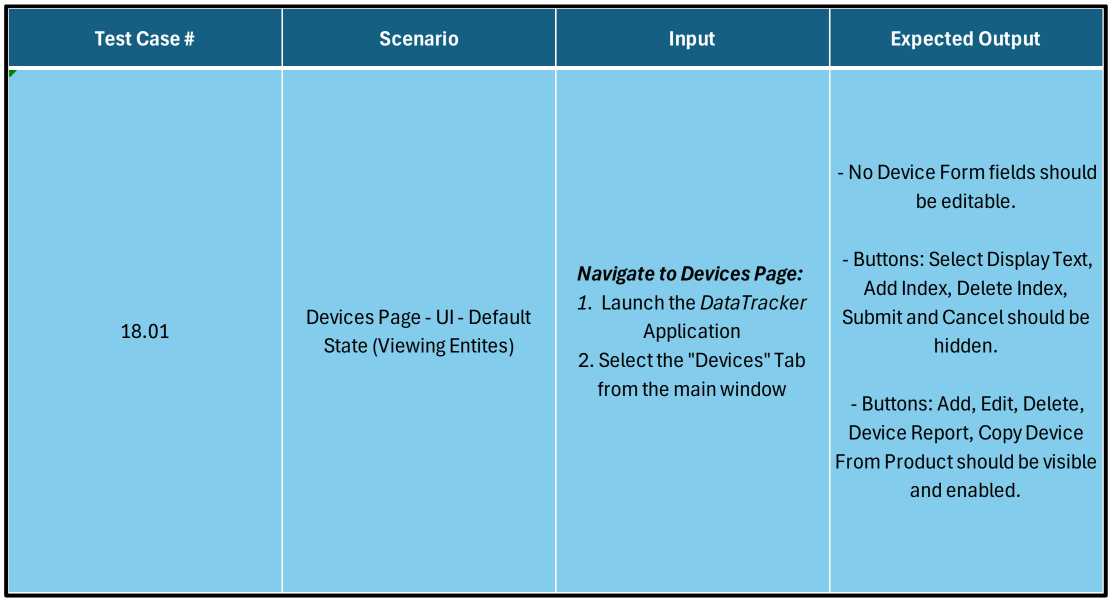
The first column of the first row in Table 2 shows user story 18, "A User can Add Devices on each Project", and the sprint tasks associated with that user story, tasks 1, 2, 3 and 4. The second column indicates where the test cases for each user story are located in the test document, in this case that is the "18. Add Devices" tab. The third column describes at a high level what was tested within that area of the document.



Sprint 3 Test Overview		
Requirement and Task #s (see Sprint 3 sprint task document)	**"Sprint3_TestResults" Tab Name**	**Description**
18. A User can Add Devices on each Project Tasks: [1-4]	18. Add Devices	Test UI, Test generate URI, Test Select Display Text, Test Input Validation, Test datastore updates
19. A User can Modify Devices on each Project Tasks: [5-6]	19. Modify Devices	Test UI, Test Input Validation, Test datastore updates
20. A User can Delete Devices on each Project Tasks: [7-9]	20. Delete Devices	Test UI, Test datastore updates
24. A User can Add Enumerations on each Project Tasks: [16 - 18]	24. Add Enumerations	Test UI, Test generate URI, Test Select Display Text, Test Input Validation, Test datastore updates
25. A User can Modify Enumerations on each project Tasks: [19-21]	25. Modify Enumerations	Test UI, Test Input Validation, Test datastore updates
26. A User can Delete Enumerations on each Project. Tasks: [22-24]	26. Delete Enumerations	Test UI, Test datastore updates
30. A User can Add an Abbreviation (Shared amoung all projects) Tasks: [10 - 12]	30. Add Abbreviations	Test UI, Test generate URI, Test Select Display Text, Test Input Validation, Test datastore updates
32. A User can Delete Abbreviations (Shared amoung all projects) Tasks: [13-15]	32. Delete Abbreviations	Test UI, Test datastore updates

Table 2: Sprint 3 Test Overview

As mentioned above, each user story has its own specific test cases. Each test case is referenced by its own unique identifier. Table 3 shows test case "18.01", indicating that it is test case 1 for user story 18. The test document page for user story 18 contains 23 separate test cases, however the same format shown in table 3 for test case "18.01" is shared across all test cases developed during this project.



Test Case #	Scenario	Input	Expected Output
18.01	Devices Page - UI - Default State (Viewing Entites)	***Navigate to Devices Page:*** *1.* Launch the *DataTracker* Application 2. Select the "Devices" Tab from the main window	- No Device Form fields should be editable. - Buttons: Select Display Text, Add Index, Delete Index, Submit and Cancel should be hidden. - Buttons: Add, Edit, Delete, Device Report, Copy Device From Product should be visible and enabled.

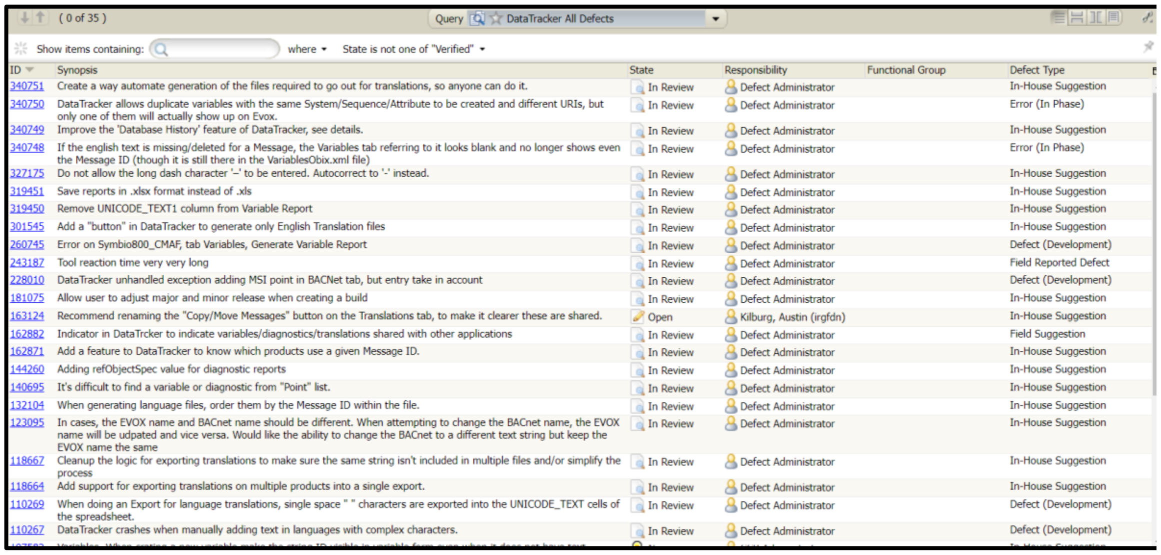
Table 3: Test Case 18.01 - Device Page - UI

Each test case is formatted with a unique identifier, a description of the scenario being tested, the user input necessary to execute the functionality under test, the expected output, the results which include a capture of what was being evaluated during the test, and finally an optional notes column to assist any future test engineer in evaluating the results of the test case. Though the results and notes columns are not shown in table 3, they are present for every test case in the appropriate sprint testing document.

The total number of development test cases generated throughout this project was 319, meaning that on average each user story had approximately 7.6 test cases created in development alongside the functionality implemented in that sprint.

The number of development test cases generated for each user story heavily depended on the complexity of the user story added. User story 20, "A User can delete Devices on each Project" for example only had three test cases. One test case ensured that a user could interact with *DataTracker's* UI as expected, allowing for the options of device deletion, or the cancellation of that deletion. The second test was used to ensure that the cancelation of a deletion would not impact the datastore. And the third test involved analyzing the artifacts impacted in the datastore to ensure that the deletion of the device was properly executed. As mentioned above, user story 18, "A User can Add Devices on each Project", had 23 separate development test cases evaluating the UI, input validation, and datastore modification.

The original *DataTracker* application had a significant backlog of defects in the Unit Controls team's defect tracking tool. These defects capture known bugs or issues in the existing application as well as potential improvements. A subset of these defects can be seen in figure 14. Each of these defects were evaluated during the design and implementation of the re-engineered *DataTracker*. Some examples of this include *Defect 243187* relating to the tool's latency and *Defect 340750* indicating that the input validation in the existing tool does not prevent multiple variables to be created with the same unique identifier. As mentioned in section 7.1 Results of Re-engineering, the existing tools latency was addressed in the architecture design of the re-engineered *DataTracker*, satisfying the needs of *Defect 243187*. *Defect 340750* was addressed in sprint 2 of this project via the requirements "A User can add Variables on each project", and "A user can Modify existing Variables on each project". During the sprint planning phase of sprint 2 the author and stakeholders discussed the requirements selected for that sprint and designed the input validation required to satisfy *Defect 340750* adding this information to the requirements specification [3].



(0 of 35)   Query   DataTracker All Defects

Show items containing: where   State is not one of "Verified"

ID	Synopsis	State	Responsibility	Functional Group	Defect Type
340751	Create a way automate generation of the files required to go out for translations, so anyone can do it.	In Review	Defect Administrator		In-House Suggestion
340750	DataTracker allows duplicate variables with the same System/Sequence/Attribute to be created and different URIs, but only one of them will actually show up on Evox.	In Review	Defect Administrator		Error (In Phase)
340749	Improve the 'Database History' feature of DataTracker, see details.	In Review	Defect Administrator		In-House Suggestion
340748	If the english text is missing/deleted for a Message, the Variables tab referring to it looks blank and no longer shows even the Message ID (though it is still there in the VariablesObix.xml file)	In Review	Defect Administrator		Error (In Phase)
327175	Do not allow the long dash character '–' to be entered. Autocorrect to '-' instead.	In Review	Defect Administrator		In-House Suggestion
319451	Save reports in .xlsx format instead of .xls	In Review	Defect Administrator		In-House Suggestion
319450	Remove UNICODE_TEXT1 column from Variable Report	In Review	Defect Administrator		In-House Suggestion
301545	Add a "button" in DataTracker to generate only English Translation files	In Review	Defect Administrator		In-House Suggestion
260745	Error on Symbio800_CMAF, tab Variables, Generate Variable Report	In Review	Defect Administrator		Defect (Development)
243187	Tool reaction time very very long	In Review	Defect Administrator		Field Reported Defect
228010	DataTracker unhandled exception adding MSI point in BACNet tab, but entry take in account	In Review	Defect Administrator		Defect (Development)
181075	Allow user to adjust major and minor release when creating a build	In Review	Defect Administrator		In-House Suggestion
163124	Recommend renaming the "Copy/Move Messages" button on the Translations tab, to make it clearer these are shared.	Open	Kilburg, Austin (irgfdn)		In-House Suggestion
162882	Indicator in DataTrcker to indicate variables/diagnostics/translations shared with other applications	In Review	Defect Administrator		Field Suggestion
162871	Add a feature to DataTracker to know which products use a given Message ID.	In Review	Defect Administrator		In-House Suggestion
144260	Adding refObjectSpec value for diagnostic reports	In Review	Defect Administrator		In-House Suggestion
140695	It's difficult to find a variable or diagnostic from "Point" list.	In Review	Defect Administrator		In-House Suggestion
132104	When generating language files, order them by the Message ID within the file.	In Review	Defect Administrator		In-House Suggestion
123095	In cases, the EVOX name and BACnet name should be different. When attempting to change the BACnet name, the EVOX name will be udpated and vice versa. Would like the ability to change the BACnet to a different text string but keep the EVOX name the same	In Review	Defect Administrator		In-House Suggestion
118667	Cleanup the logic for exporting translations to make sure the same string isn't included in multiple files and/or simplify the process	In Review	Defect Administrator		In-House Suggestion
118664	Add support for exporting translations on multiple products into a single export.	In Review	Defect Administrator		In-House Suggestion
110269	When doing an Export for language translations, single space " " characters are exported into the UNICODE_TEXT cells of the spreadsheet.	In Review	Defect Administrator		Defect (Development)
110267	DataTracker crashes when manually adding text in languages with complex characters.	In Review	Defect Administrator		Defect (Development)

Figure 14: Original *DataTracker* Defect Backlog

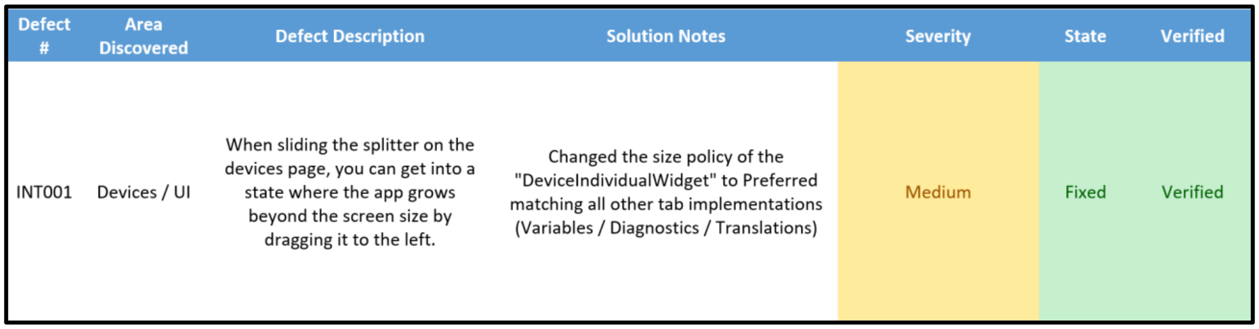
It is important to note that not all defects from the original *DataTracker* would be applicable to the re-engineered *DataTracker*. For example, *Defect 118664* captures a request for exporting translations into a single file. This functionality is expected to be implemented in phase 2 of the *DataTracker* re-engineering project. Another example, *Defect 140695* captures the difficulty that users encounter when attempting to find a point's associated variable. This defect is not applicable to the re-engineered *DataTracker* as points will not be supported in the tool. Points are created using a separate tool on the newest embedded application controller platform. These examples are highlighted to show that not all defects from the original tool are applicable to the new system.

From the backlog of 44 defects against the existing *DataTracker* application, 11 were related to functionality that will not be supported in the re-engineered *DataTracker*. Ten defects are applicable to functionality that will be added in phase 2 of the re-engineering project. The remaining 23 defects were applicable to the functionality implemented in phase 1 of the re-engineering project defined in this manuscript. All 23 applicable defects were tested against the re-engineered *DataTracker* as part of development to ensure that every documented issue within the existing *DataTracker* was addressed.

After the initial development of the re-engineered *DataTracker* had been completed, the author performed integration testing. Integration testing of the re-engineered *DataTracker* included performing exploratory testing, and regression testing. Exploratory testing involved following typical use cases that involve adding, modifying or deleting many entities across the DataTracker application. An example of this is following the necessary steps to add the Variables, Diagnostics, Devices, and Translations required for an embedded chiller application to contain the information necessary to support a new device. This required verifying that the UI was updated successfully from the datastore as entities were updated, and that the tool behaved as expected when exercising many of its functionalities in more typical use cases.

Regression testing included re-evaluating the 319 developer test cases generated during development, as well as re-evaluating the backlog of defects written against the existing *DataTracker* application. The goal of this testing was to ensure that there was no unintended behavior caused by the addition of functionality in later sprints, ensuring that the tool behaved the same way that it had during the stage in development when that functionality was originally added.

Through integration testing, the author could be confident that the re-engineered *DataTracker* system was functioning correctly after each feature of the application had been developed and tested individually. Through this style of testing the author discovered a handful of defects that were introduced in development. All defects discovered in this effort were recorded in an integration testing document that captured a unique identifier, area discovered, defect description, solution notes, severity, state, and the status of verification for the defect's solution.



Defect #	Area Discovered	Defect Description	Solution Notes	Severity	State	Verified
INT001	Devices / UI	When sliding the splitter on the devices page, you can get into a state where the app grows beyond the screen size by dragging it to the left.	Changed the size policy of the "DeviceIndividualWidget" to Preferred matching all other tab implementations (Variables / Diagnostics / Translations)	Medium	Fixed	Verified

Figure 15: Integration Testing *Defect INT001*

An example of one of these defects, *Defect INT001*, can be seen in figure 15. All defects captured during integration testing utilized the prefix of *INT*, short for integration, to avoid any confusion with the defect numbers utilized by the Unit Controls Team's defect tracking tool. *Defect INT001* captures a defect in which a UI element when dragged or manipulated by a user, unexpectedly expands the main window of the application. *Defect INT001* was found through the regression testing performed on the developer test cases created during the development of sprint 1. The author was able to correct this issue as well as many others that would likely have only been discovered through this style of testing.

All test cases identified during development and integration testing were compiled into a single test document. This document encapsulates the test suite for the re-engineered *DataTracker* application, serving as a comprehensive resource for future testing [5].

7. Conclusion

This project has been a fantastic opportunity to apply the knowledge that the author has gained through the Master of Software Engineering program at the University of Wisconsin – La Crosse. The author was able to lead a software development project utilizing the Agile approach from start to finish with a real software development team.

This project allowed the author to work through the challenges of understanding a large existing system, and re-engineering that system based on the needs of a real-world customer. The re-engineered *DataTracker* will be developed further in phase 2 of the project before its eventual deployment and replacement of the existing *DataTracker*.

7.1 Result of Re-engineering

It can be said that this initial phase of the *DataTracker* re-engineering project was a success. Almost all base functionalities of the original *DataTracker* application have been captured, along with improvements to tool latency, maintainability, and usability. Additionally, this effort enabled the team to address multiple user pain points observed when using the original tool.

Through the redesign of the application's architecture, the latency that was experienced with the original *DataTracker* has been significantly improved. By leveraging the existing version control system, and not an outdated, cluttered, Oracle database served out of one location, the latency seen in the original tool will not be experienced. In other words, it does not matter where the users of the application are located geographically, the re-engineered *DataTracker* will operate as quickly as the user's computer allows. This is because the new application utilizes file I/O with locally checked out files from the existing version control system. An example of this improved latency can be seen when accessing the translation page of both *DataTracker* applications. This page populates almost instantaneously in the re-engineered application, whereas in the past, depending on the user's geographic location, this could take upwards of 5 minutes.

This project also enabled an improvement to maintainability through a collaborative design process, and the use of a well-known design pattern. Using the Agile approach to manage this project enabled the ongoing collaboration between the author and *DataTracker's* stakeholders. This allowed multiple members of the Unit Controls team to be directly involved in the design and development of the re-engineered application, fostering a greater understanding of the entities managed by *DataTracker* and their relationships.

By utilizing the Model View Controller (MVC) design pattern, developers can more accurately discern the location of specific functionality within the re-engineered application. The use of this design pattern also enables application improvements or modifications to specific components of the application, with mitigated risk to other areas.

Beyond just the implementation of the *DataTracker* application, the datastore component of the overall *DataTracker* system was significantly simplified. As the original system matured, a large amount of its functionalities were no longer needed by its users. Most of these functionalities were implemented to support older controller hardware generations in which the architecture of the embedded chiller application build machine was significantly different. The existing *DataTracker* application also supported user role and user access functionality, which in the re-engineered *DataTracker* is provided by the existing version control system. Through this design decision a significant amount of the existing datastore could be simplified or eliminated. As mentioned in section 4.4 Datastore Design, this allowed the re-engineered datastore to reduce the number of entities required from 77 to 16.

Maintainability of the re-engineered *DataTracker* should also see an improvement through the documentation created as part of this project. As mentioned in section 1.1 Background, the original *DataTracker* application had no documentation pertaining to the design or implementation of the tool. From the requirement and testing documents, as well as the many diagrams and illustrations the author has created, the author plans to maintain a user manual for the re-engineered *DataTracker*.

This re-engineering effort also targeted an improvement to the usability of *DataTracker*. Many of these design choices can be seen in the Appendix of this manuscript. An obvious example of this improvement is the addition of search functionality to all main entities in the application.

With the team addressing the non-functional requirements, functional requirements, and all applicable defects in the existing *DataTracker* application's backlog, the re-engineering project was an overall success. The new *DataTracker* included all necessary base functionality of the existing tool while improving upon its latency, maintainability, usability, and known issues or pain points documented over the original tools almost 20-year lifespan.

7.2 Challenges

Many challenges were faced during the re-engineering of *DataTracker*. Several of these challenges include the lack of documentation and ownership of the existing *DataTracker*, the author's lack of familiarity with the PyQt framework and Qt Designer, and ensuring that the application produced satisfied the needs of *DataTracker's* stakeholders while keeping the project on schedule.

The lack of documentation and ownership of the existing *DataTracker* application made it difficult to identify the details necessary to design the re-engineered solution relating to each user story. The author addressed this lack of information by investigating the existing implementation related to the targeted user stories of each sprint prior to the requirement analysis phase. The author would identify the existing entities and their relationships in the original *DataTracker* by evaluating the implementation of the tool and its related Oracle database. The author would additionally identify how to output the necessary test data from the original database to ensure that the functionality added in that sprint could be adequately tested.

This analysis allowed the author and stakeholders to discuss the current implementation in terms of what needed to be supported and what improvements could be made. As mentioned in section 2.2 Model Used – Agile, the Agile approach was selected for this project to mitigate this lack of well-defined requirements early in the project. This approach allowed for the author and stakeholders to refine the functional requirements as more information was gathered in each sprint.

Another challenge faced during this project was the author’s lack of familiarity with PyQt applications and the Qt Designer tool. In section 4.2 Technology Selection, the author and stakeholders selected the PyQt framework, even though the author had no prior experience implementing a desktop application with these technologies. The author created several simple experimental applications in PyQt with Qt Designer to further familiarize themselves with some of the nuances of the framework and assist in the selection of the design pattern that would be utilized by the re-engineered *DataTracker*. This challenge was also addressed through the author’s research of tutorials and the framework’s documentation to better understand the abilities and shortcomings of writing software using these tools. As the project progressed, the author gained a stronger understanding of the framework and UI design tool. This led to several areas of the application being re-written. The most notable example of this is the replacement of many of the form field’s text entry elements being updated to use a more appropriate type in Qt Designer, to prevent unintended UI behavior.

Part of the effort necessary to ensure that the application met the needs of *DataTracker*’s stakeholders was documenting and addressing feedback given during the demonstrations of the functionality added during each sprint. Deciding whether to take on all feedback during the current sprint or to document the feedback for phase 2 of this project was a challenge. The author worked with the stakeholders to prioritize and evaluate the time and complexity needed to address the feedback given during these demonstrations. Due to the aggressive timeline associated with this project, the author ensured that the project schedule could handle these changes. Through this effort, much of the feedback was addressed as part of this project, however lower priority changes were captured in a backlog of potential improvements gathered for phase 2 of this project. By evaluating what feedback was most important to *DataTracker’s* stakeholders, and addressing that feedback during this project, the author ensured that the customer would be satisfied with the end product.

A challenge was also seen early in the project when the author decided to include approximately 15 software professionals from the Unit Controls team in the architecture and technology selection process. Although it was important to have buy-in from the extended Unit Controls team for these discussions, the author felt that having this many stakeholders in each sprint planning and retrospective meeting would slow the progress of the project. The author worked with the extended group of stakeholders to identify a smaller team of representatives to take part in the sprint planning and retrospective meetings associated with each sprint of the project. This allowed for these meetings to operate in a more timely manner ensuring that the project remained on schedule.

7.3 Future Work

Due to the size of the original *DataTracker* application, the re-engineering effort has been divided into multiple phases. Phase 1, the project described in this manuscript, would capture as much of the functionality of the original application as possible, targeting the most important functionality first. Phase 2 would capture added improvements, and any remaining functionality not addressed in the first phase of the overall re-engineering effort.

Some of the requirements to be addressed in phase 2 of the *DataTracker* re-engineering effort include adding, modifying, and deleting LLID profiles, Scalar Unit Types, and Operating modes. These areas were identified for phase 2 because they are areas of low change in the datastore and are very infrequently updated within the current *DataTracker* system.

Additional phase 2 requirements capture the functionality of exporting translations into an Excel format to be utilized by a translation company, and the import of those updated files back into the *DataTracker* application in batches. This was targeted for phase 2 of this project as the author and stakeholders agreed that a larger discussion related to the design of this feature would need to take place with the group of extended stakeholders before this could be implemented. Due to the importance of this feature's design, it will be the primary focus of phase 2. The scope of this feature was significant enough that the author and stakeholders felt that it was not possible to complete within the timeline of phase 1 of this project.

In phase 2 of this re-engineering effort the author will work with a team of 2 software engineering interns to implement the functionality mentioned above over the next calendar year. Phase 2 of this effort will operate in a similar method to phase 1, again following the Agile approach. Additionally, the author will work with *DataTracker*'s stakeholders to solidify a deployment strategy that minimizes disruption from the user perspective.

7.5 Deployment

7.5.1 Deployment Strategy

The strategy for deployment of the re-engineered *DataTracker* will heavily depend on the translation and display text entities implemented in *DataTracker's* datastore. As shown in figure 8 in section 4.4 Datastore Design, the translation entity has a relationship with most entities in the datastore. The translation entity is also shared across all embedded chiller products supported by the *DataTracker* system. This would make an incremental deployment, where each embedded chiller project would slowly begin using the new *DataTracker* system unlikely as the translation entity's primary key, message ID, would need to remain unique across both the existing Oracle database and the new version controlled datastore.

There are ways to get around this issue, such as ensuring that translations created in the re-engineered system start at an offset providing separation between the two datastores and ensuring that the same message ID is not referencing multiple translation entities once that data is imported into the re-engineered system. This of course would add more complexity to this transition, including the potential for duplicated translated text. A similar strategy would need to be applied to the display text entity as well.

It is for these reasons that a complete transition from the existing *DataTracker* system to the re-engineered system during a scheduled deployment period is far more likely. During the development of this project the author has documented the necessary interactions with the existing *DataTracker* database to export and format all files needed by the re-engineered *DataTracker*. These procedures would need to take place for each project that the re-engineered solution would need to support. Luckily, the list of approximately 20 embedded chiller applications can be prioritized based on which projects are actually active, decreasing the number of times this procedure would need to be followed for the initial deployment of this tool. The author and team could then incrementally generate the necessary files for the inactive projects after the initial deployment of the re-engineered system has taken place.

The original *DataTracker* will remain active for older controller hardware generations only. As mentioned in section 4.4 Datastore Design, the original application will support functionalities needed by the older unique build machines for these products. The original *DataTracker* will remain active until the Unit Controls team decides to no longer support service pack releases for these products. No active development or improvement projects currently take place on these products; only necessary fixes to correct issues reported from the field are made, meaning that the datastore will be changed infrequently.

7.5.2 Application Deployment Method

The method of deploying the re-engineered *DataTracker* application could be accomplished using PyInstaller and Inno Setup. PyInstaller is a tool used to convert Python applications and their dependencies into one standalone executable. Utilizing PyInstaller would allow for users of the application to avoid installing a specific Python interpreter or the modules the re-engineered *DataTracker* application depends on. This also ensures that all users are using the same Python interpreter and version of the modules utilized by the application. Inno Setup is a script-driven installation system that is used to create Windows installers. Inno Setup supports a customizable setup UI, an application installer, and an application uninstaller if properly configured.

Using PyInstaller to create the *DataTracker* executable and Inno Setup to create the installer would assist in the distribution of not only the executable, but also the necessary configuration files used by *DataTracker*. These configuration files are primarily used to store the selected theme for *DataTracker's* UI as well as the paths to the locally checked out files from the version controlled datastore.

Updating the application as new revisions are released could be managed in several ways. The most likely way at the time of writing this manuscript would be to potentially leverage an update mechanism within the *DataTracker* application that will check for an updated installation file in a location on the Trane Technologies network. If the installation file is newer than the currently installed version, then an automatic update would be performed. Should the installation file location not be available then the software would continue to operate as though no updated installation file existed.

8. Bibliography

[1] "Software Development Resources | Qt," www.qt.io, 2017. https://www.qt.io/resources/qt?content-type=Success+Story (accessed Mar. 02, 2025).

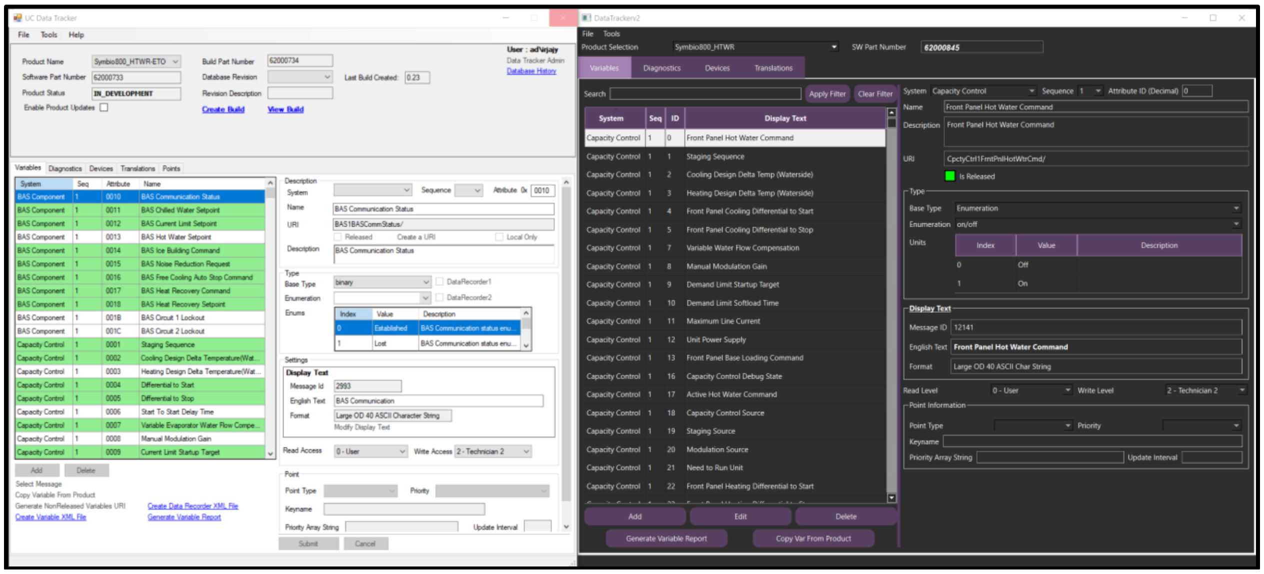
[2] R. Python, "Python and PyQt: Building a GUI Desktop Calculator – Real Python," realpython.com. https://realpython.com/python-pyqt-gui-calculator/

[3] Miller, Graham. "Software Requirements Document for DataTracker", Version 1.1, April 2025.

[4] Miller, Graham. "Software Design Document for DataTracker", Version 1.1, April 2025.

[5] Miller, Graham. "Software Test Suite for DataTracker", Version 1.1, April 2025.

Appendix: GUI Before and After



UC Data Tracker

User : ad\vrjajy
Data Tracker Admin
Database History

Product Name	Symbio800_HTWR-ETO	Build Part Number	62000734
Software Part Number	62000733	Database Revision	
Product Status	IN_DEVELOPMENT	Revision Description	
Enable Product Updates	☐	Last Build Created:	0.23

Create Build · View Build

System	Seq	Attribute	Name
BAS Component	1	0010	BAS Communication Status
BAS Component	1	0011	BAS Chilled Water Setpoint
BAS Component	1	0012	BAS Current Limit Setpoint
BAS Component	1	0013	BAS Hot Water Setpoint
BAS Component	1	0014	BAS Ice Building Command
BAS Component	1	0015	BAS Noise Reduction Request
BAS Component	1	0016	BAS Free Cooling Auto Stop Command
BAS Component	1	0017	BAS Heat Recovery Command
BAS Component	1	0018	BAS Heat Recovery Setpoint
BAS Component	1	001B	BAS Circuit 1 Lockout
BAS Component	1	001C	BAS Circuit 2 Lockout
Capacity Control	1	0001	Staging Sequence
Capacity Control	1	0002	Cooling Design Delta Temperature(Wat...
Capacity Control	1	0003	Heating Design Delta Temperature(Wat...
Capacity Control	1	0004	Differential to Start
Capacity Control	1	0005	Differential to Stop
Capacity Control	1	0006	Start To Start Delay Time
Capacity Control	1	0007	Variable Evaporator Water Flow Compe...
Capacity Control	1	0008	Manual Modulation Gain
Capacity Control	1	0009	Current Limit Startup Target

Description — System: ; Sequence: ; Attribute 0x: 0010
Name: BAS Communication Status
URI: BAS1BASCommStatus/ ☐ Released Create a URI ☐ Local Only
Description: BAS Communication Status

Type — Base Type: binary ☐ DataRecorder1; Enumeration: ☐ DataRecorder2

Index	Value	Description
0	Established	BAS Communication status enu...
1	Lost	BAS Communication status enu...

Settings — Display Text
Message Id: 2993
English Text: BAS Communication
Format: Large OD 40 ASCII Character String
Modify Display Text
Read Access: 0 - User · Write Access: 2 - Technician 2

Point — Point Type: · Priority: · Keyname: · Priority Array String: · Update Interval:

Select Message
Copy Variable From Product
Generate NonReleased Variables URI
Create Variable XML File
Create Data Recorder XML File
Generate Variable Report

DataTrackerv2

Product Selection: Symbio800_HTWR · SW Part Number: 62000845

System	Seq	ID	Display Text
Capacity Control	1	0	Front Panel Hot Water Command
Capacity Control	1	1	Staging Sequence
Capacity Control	1	2	Cooling Design Delta Temp (Waterside)
Capacity Control	1	3	Heating Design Delta Temp (Waterside)
Capacity Control	1	4	Front Panel Cooling Differential to Start
Capacity Control	1	5	Front Panel Cooling Differential to Stop
Capacity Control	1	7	Variable Water Flow Compensation
Capacity Control	1	8	Manual Modulation Gain
Capacity Control	1	9	Demand Limit Startup Target
Capacity Control	1	10	Demand Limit Softload Time
Capacity Control	1	11	Maximum Line Current
Capacity Control	1	12	Unit Power Supply
Capacity Control	1	13	Front Panel Base Loading Command
Capacity Control	1	16	Capacity Control Debug State
Capacity Control	1	17	Active Hot Water Command
Capacity Control	1	18	Capacity Control Source
Capacity Control	1	19	Staging Source
Capacity Control	1	20	Modulation Source
Capacity Control	1	21	Need to Run Unit
Capacity Control	1	22	Front Panel Heating Differential to Start

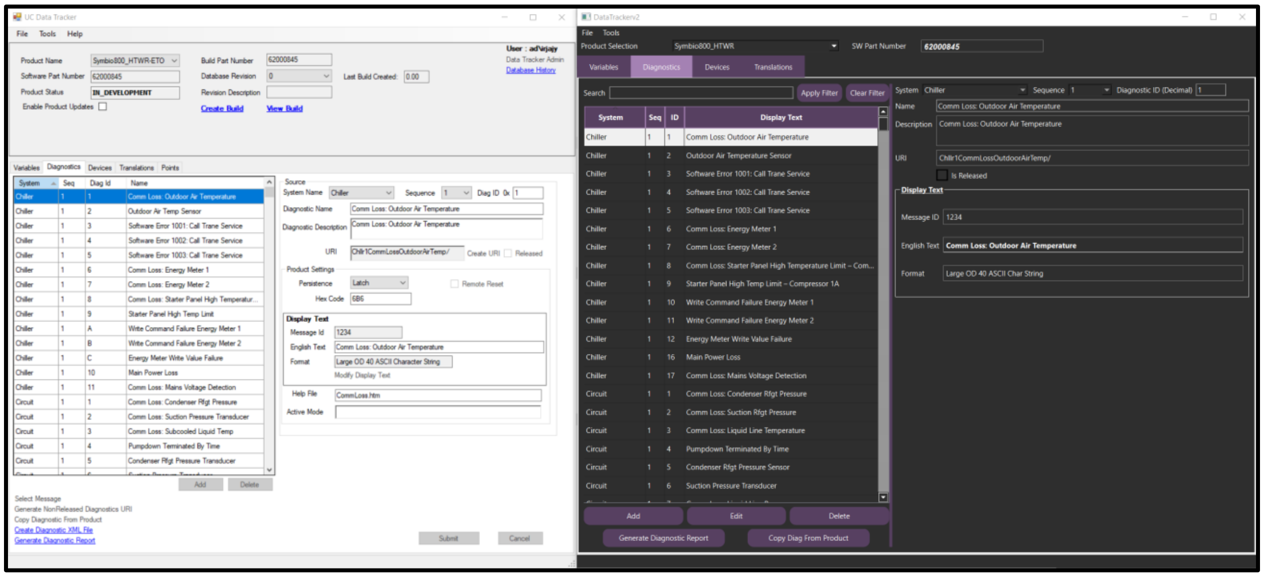
System: Capacity Control · Sequence: 1 · Attribute ID (Decimal): 0
Name: Front Panel Hot Water Command
Description: Front Panel Hot Water Command
URI: CpctyCtrl1FrntPnlHotWtrCmd/
🟩 Is Released

Type — Base Type: Enumeration; Enumeration: on/off

Index	Value	Description
0	Off	
1	On	

Display Text — Message ID: 12141; English Text: **Front Panel Hot Water Command**; Format: Large OD 40 ASCII Char String
Read Level: 0 - User · Write Level: 2 - Technician 2
Point Information — Point Type: · Priority: · Keyname: · Priority Array String: · Update Interval:

Figure 16: *DataTracker* Variables Page Before and After



UC Data Tracker

User : ad\vrjajy
Data Tracker Admin
Database History

Product Name	Symbio800_HTWR-ETO	Build Part Number	62000845
Software Part Number	62000845	Database Revision	0
Product Status	IN_DEVELOPMENT	Revision Description	
Enable Product Updates	☐	Last Build Created:	0.00

Create Build · View Build

System	Seq	Diag Id	Name
Chiller	1	1	Comm Loss: Outdoor Air Temperature
Chiller	1	2	Outdoor Air Temp Sensor
Chiller	1	3	Software Error 1001: Call Trane Service
Chiller	1	4	Software Error 1002: Call Trane Service
Chiller	1	5	Software Error 1003: Call Trane Service
Chiller	1	6	Comm Loss: Energy Meter 1
Chiller	1	7	Comm Loss: Energy Meter 2
Chiller	1	8	Comm Loss: Starter Panel High Temperatur...
Chiller	1	9	Starter Panel High Temp Limit
Chiller	1	A	Write Command Failure Energy Meter 1
Chiller	1	B	Write Command Failure Energy Meter 2
Chiller	1	C	Energy Meter Write Value Failure
Chiller	1	10	Main Power Loss
Chiller	1	11	Comm Loss: Mains Voltage Detection
Circuit	1	1	Comm Loss: Condenser Rfgt Pressure
Circuit	1	2	Comm Loss: Suction Pressure Transducer
Circuit	1	3	Comm Loss: Subcooled Liquid Temp
Circuit	1	4	Pumpdown Terminated By Time
Circuit	1	5	Condenser Rfgt Pressure Transducer

Source — System Name: Chiller · Sequence: 1 · Diag ID 0x: 1
Diagnostic Name: Comm Loss: Outdoor Air Temperature
Diagnostic Description: Comm Loss: Outdoor Air Temperature
URI: Chllr1CommLossOutdoorAirTemp/ Create URI ☐ Released

Product Settings — Persistence: Latch ☐ Remote Reset; Hex Code: 6B6

Display Text — Message Id: 1234; English Text: Comm Loss: Outdoor Air Temperature; Format: Large OD 40 ASCII Character String
Modify Display Text
Help File: CommLoss.htm
Active Mode:

Select Message
Generate NonReleased Diagnostics URI
Copy Diagnostic From Product
Create Diagnostic XML File
Generate Diagnostic Report

DataTrackerv2

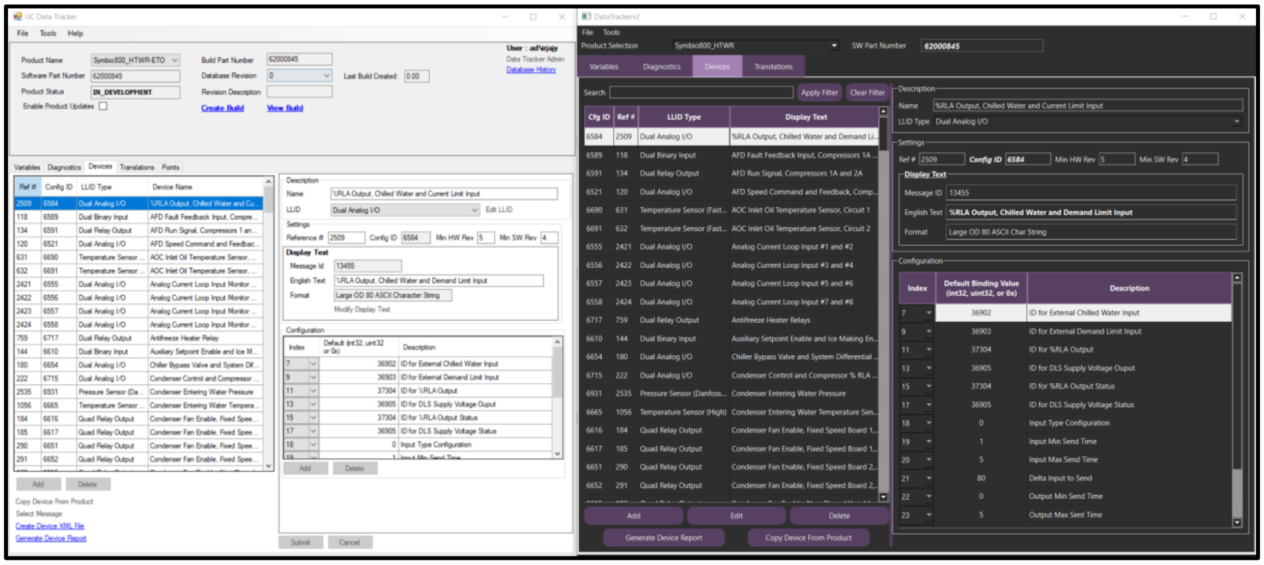
Product Selection: Symbio800_HTWR · SW Part Number: 62000845

System	Seq	ID	Display Text
Chiller	1	1	Comm Loss: Outdoor Air Temperature
Chiller	1	2	Outdoor Air Temperature Sensor
Chiller	1	3	Software Error 1001: Call Trane Service
Chiller	1	4	Software Error 1002: Call Trane Service
Chiller	1	5	Software Error 1003: Call Trane Service
Chiller	1	6	Comm Loss: Energy Meter 1
Chiller	1	7	Comm Loss: Energy Meter 2
Chiller	1	8	Comm Loss: Starter Panel High Temperature Limit – Com...
Chiller	1	9	Starter Panel High Temp Limit – Compressor 1A
Chiller	1	10	Write Command Failure Energy Meter 1
Chiller	1	11	Write Command Failure Energy Meter 2
Chiller	1	12	Energy Meter Write Value Failure
Chiller	1	16	Main Power Loss
Chiller	1	17	Comm Loss: Mains Voltage Detection
Circuit	1	1	Comm Loss: Condenser Rfgt Pressure
Circuit	1	2	Comm Loss: Suction Rfgt Pressure
Circuit	1	3	Comm Loss: Liquid Line Temperature
Circuit	1	4	Pumpdown Terminated By Time
Circuit	1	5	Condenser Rfgt Pressure Sensor
Circuit	1	6	Suction Pressure Transducer

System: Chiller · Sequence: 1 · Diagnostic ID (Decimal): 1
Name: Comm Loss: Outdoor Air Temperature
Description: Comm Loss: Outdoor Air Temperature
URI: Chllr1CommLossOutdoorAirTemp/
☐ Is Released

Display Text — Message ID: 1234; English Text: **Comm Loss: Outdoor Air Temperature**; Format: Large OD 40 ASCII Char String

Figure 17: *DataTracker* Diagnostics Page Before and After



User : ad\vrjajy
Data Tracker Admin

Product Name: Symbio800_HTWR-ETO
Software Part Number: 62000845
Product Status: IN_DEVELOPMENT
Enable Product Updates: ☐
Build Part Number: 62000845
Database Revision: 0
Revision Description:
Last Build Created: 0.00

Ref #	Config ID	LLID Type	Device Name
2509	6584	Dual Analog I/O	%RLA Output, Chilled Water and Cu...
118	6589	Dual Binary Input	AFD Fault Feedback Input, Compre...
134	6591	Dual Relay Output	AFD Run Signal, Compressors 1 an...
120	6521	Dual Analog I/O	AFD Speed Command and Feedbac...
631	6690	Temperature Sensor ...	AOC Inlet Oil Temperature Sensor, ...
632	6691	Temperature Sensor ...	AOC Inlet Oil Temperature Sensor, ...
2421	6555	Dual Analog I/O	Analog Current Loop Input Monitor ...
2422	6556	Dual Analog I/O	Analog Current Loop Input Monitor ...
2423	6557	Dual Analog I/O	Analog Current Loop Input Monitor ...
2424	6558	Dual Analog I/O	Analog Current Loop Input Monitor ...
759	6717	Dual Relay Output	Antifreeze Heater Relay
144	6610	Dual Binary Input	Auxiliary Setpoint Enable and Ice M...
180	6654	Dual Analog I/O	Chiller Bypass Valve and System Dif...
222	6715	Dual Analog I/O	Condenser Control and Compressor ...
2535	6931	Pressure Sensor (Da...	Condenser Entering Water Pressure
1056	6665	Temperature Sensor ...	Condenser Entering Water Tempera...
184	6616	Quad Relay Output	Condenser Fan Enable, Fixed Spee...
185	6617	Quad Relay Output	Condenser Fan Enable, Fixed Spee...
290	6651	Quad Relay Output	Condenser Fan Enable, Fixed Spee...
291	6652	Quad Relay Output	Condenser Fan Enable, Fixed Spee...

Description
Name: %RLA Output, Chilled Water and Current Limit Input
LLID: Dual Analog I/O
Settings
Reference #: 2509 Config ID: 6584 Min HW Rev: 5 Min SW Rev: 4
Display Text
Message Id: 13455
English Text: %RLA Output, Chilled Water and Demand Limit Input
Format: Large OD 80 ASCII Character String

Configuration

Index	Default (int32, uint32 or 0x)	Description
7	36902	ID for External Chilled Water Input
9	36903	ID for External Demand Limit Input
11	37304	ID for %RLA Output
13	36905	ID for DLS Supply Voltage Ouput
15	37304	ID for %RLA Output Status
17	36905	ID for DLS Supply Voltage Status
18	0	Input Type Configuration

Product Selection: Symbio800_HTWR
SW Part Number: 62000845
Search:

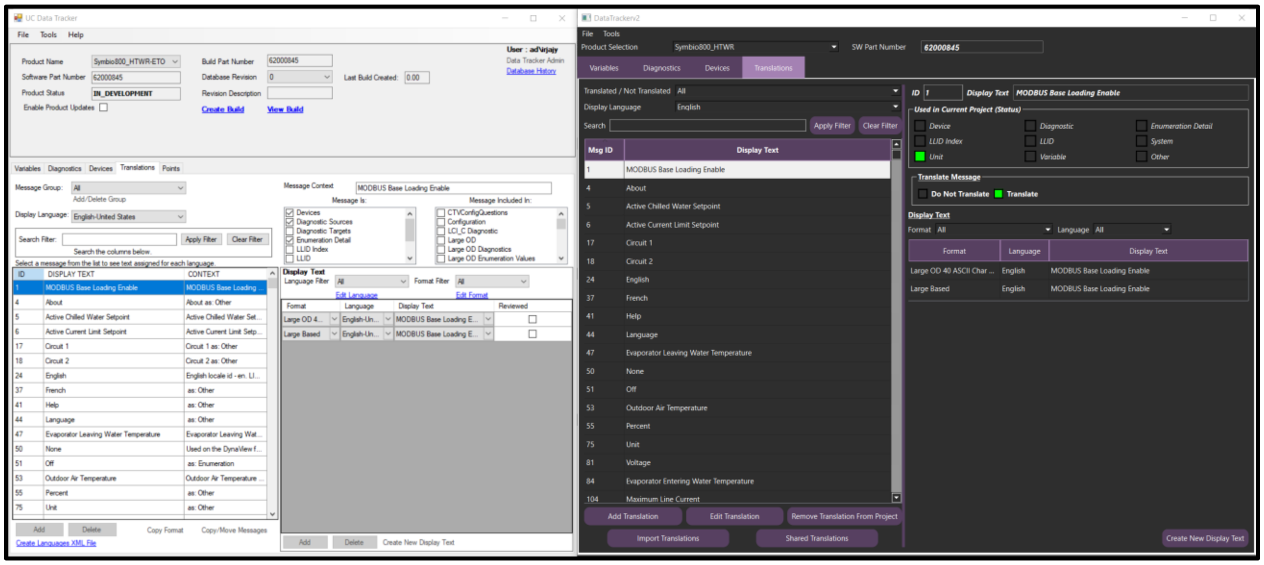
Cfg ID	Ref #	LLID Type	Display Text
6584	2509	Dual Analog I/O	%RLA Output, Chilled Water and Demand Li...
6589	118	Dual Binary Input	AFD Fault Feedback Input, Compressors 1A ...
6591	134	Dual Relay Output	AFD Run Signal, Compressors 1A and 2A
6521	120	Dual Analog I/O	AFD Speed Command and Feedback, Comp...
6690	631	Temperature Sensor (Fast...	AOC Inlet Oil Temperature Sensor, Circuit 1
6691	632	Temperature Sensor (Fast...	AOC Inlet Oil Temperature Sensor, Circuit 2
6555	2421	Dual Analog I/O	Analog Current Loop Input #1 and #2
6556	2422	Dual Analog I/O	Analog Current Loop Input #3 and #4
6557	2423	Dual Analog I/O	Analog Current Loop Input #5 and #6
6558	2424	Dual Analog I/O	Analog Current Loop Input #7 and #8
6717	759	Dual Relay Output	Antifreeze Heater Relays
6610	144	Dual Binary Input	Auxiliary Setpoint Enable and Ice Making En...
6654	180	Dual Analog I/O	Chiller Bypass Valve and System Differential ...
6715	222	Dual Analog I/O	Condenser Control and Compressor % RLA ...
6931	2535	Pressure Sensor (Danfoss...	Condenser Entering Water Pressure
6665	1056	Temperature Sensor (High)	Condenser Entering Water Temperature Sen...
6616	184	Quad Relay Output	Condenser Fan Enable, Fixed Speed Board 1,...
6617	185	Quad Relay Output	Condenser Fan Enable, Fixed Speed Board 1,...
6651	290	Quad Relay Output	Condenser Fan Enable, Fixed Speed Board 2,...
6652	291	Quad Relay Output	Condenser Fan Enable, Fixed Speed Board 2,...

Description
Name: %RLA Output, Chilled Water and Current Limit Input
LLID Type: Dual Analog I/O
Settings
Ref #: 2509 Config ID: 6584 Min HW Rev: 5 Min SW Rev: 4
Display Text
Message ID: 13455
English Text: %RLA Output, Chilled Water and Demand Limit Input
Format: Large OD 80 ASCII Char String

Configuration

Index	Default Binding Value (int32, uint32, or 0x)	Description
7	36902	ID for External Chilled Water Input
9	36903	ID for External Demand Limit Input
11	37304	ID for %RLA Output
13	36905	ID for DLS Supply Voltage Ouput
15	37304	ID for %RLA Output Status
17	36905	ID for DLS Supply Voltage Status
18	0	Input Type Configuration
19	1	Input Min Send Time
20	5	Input Max Send Time
21	80	Delta Input to Send
22	0	Output Min Send Time
23	5	Output Max Sent Time

Figure 18: *DataTracker* Devices Page Before and After



User : ad\vrjajy
Data Tracker Admin

Product Name: Symbio800_HTWR-ETO
Software Part Number: 62000845
Product Status: IN_DEVELOPMENT
Enable Product Updates: ☐
Build Part Number: 62000845
Database Revision: 0
Revision Description:
Last Build Created: 0.00

Message Group: All
Display Language: English-United States
Search Filter:
Search the columns below.
Select a message from the list to see text assigned for each language.

ID	DISPLAY TEXT	CONTEXT
1	MODBUS Base Loading Enable	MODBUS Base Loading ...
4	About	About as: Other
5	Active Chilled Water Setpoint	Active Chilled Water Set...
6	Active Current Limit Setpoint	Active Current Limit Setp...
17	Circuit 1	Circuit 1 as: Other
18	Circuit 2	Circuit 2 as: Other
24	English	English locale id - en. LI...
37	French	as: Other
41	Help	as: Other
44	Language	as: Other
47	Evaporator Leaving Water Temperature	Evaporator Leaving Wat...
50	None	Used on the DynaView f...
51	Off	as: Enumeration
53	Outdoor Air Temperature	Outdoor Air Temperature ...
55	Percent	as: Other
75	Unit	as: Other

Message Context: MODBUS Base Loading Enable

Message Is:
- ✓ Devices
- ✓ Diagnostic Sources
- ☐ Diagnostic Targets
- ✓ Enumeration Detail
- ☐ LLID Index
- ☐ LLID

Message Included In:
- ☐ CTVConfigQuestions
- ☐ Configuration
- ☐ LCI_C Diagnostic
- ☐ Large OD
- ☐ Large OD Diagnostics
- ☐ Large OD Enumeration Values

Display Text
Language Filter: All Format Filter: All

Format	Language	Display Text	Reviewed
Large OD 4...	English-Un...	MODBUS Base Loading E...	☐
Large Based	English-Un...	MODBUS Base Loading E...	☐

Product Selection: Symbio800_HTWR
SW Part Number: 62000845
Translated / Not Translated: All
Display Language: English
Search:

Msg ID	Display Text
1	MODBUS Base Loading Enable
4	About
5	Active Chilled Water Setpoint
6	Active Current Limit Setpoint
17	Circuit 1
18	Circuit 2
24	English
37	French
41	Help
44	Language
47	Evaporator Leaving Water Temperature
50	None
51	Off
53	Outdoor Air Temperature
55	Percent
75	Unit
81	Voltage
84	Evaporator Entering Water Temperature
104	Maximum Line Current

ID: 1 Display Text: MODBUS Base Loading Enable

Used in Current Project (Status)

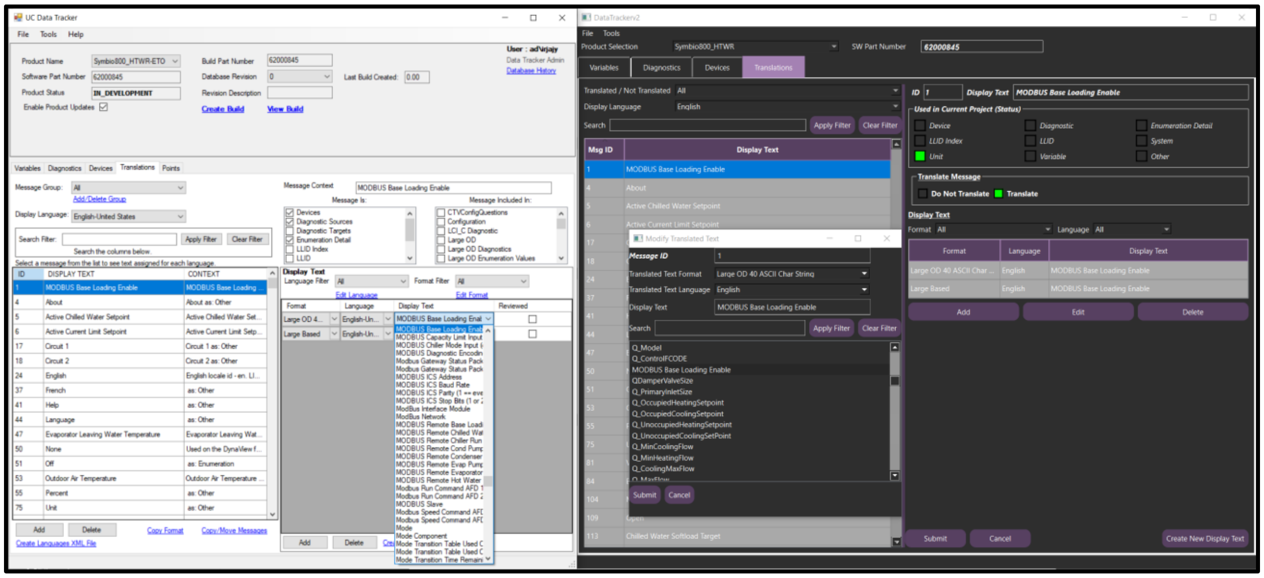
☐ Device	☐ Diagnostic	☐ Enumeration Detail
☐ LLID Index	☐ LLID	☐ System
🟩 Unit	☐ Variable	☐ Other

Translate Message
☐ Do Not Translate 🟩 Translate

Display Text
Format: All Language: All

Format	Language	Display Text
Large OD 40 ASCII Char ...	English	MODBUS Base Loading Enable
Large Based	English	MODBUS Base Loading Enable

Figure 19: *DataTracker* Translations Page Before and After



UC Data Tracker

File Tools Help

Product Name: Symbio800_HTWR-ETO
Software Part Number: 62000845
Product Status: IN_DEVELOPMENT
Enable Product Updates ☑
Build Part Number: 62000845
Database Revision: 0
Revision Description:
Last Build Created: 0.00
Create Build View Build
User : ad\rjajy
Data Tracker Admin
Database History

Variables Diagnostics Devices Translations Points

Message Group: All
Add/Delete Group
Display Language: English-United States
Search Filter: Apply Filter Clear Filter
Search the columns below.
Select a message from the list to see text assigned for each language.

ID	DISPLAY TEXT	CONTEXT
1	MODBUS Base Loading Enable	MODBUS Base Loading ...
4	About	About as: Other
5	Active Chilled Water Setpoint	Active Chilled Water Set...
6	Active Current Limit Setpoint	Active Current Limit Setp...
17	Circuit 1	Circuit 1 as: Other
18	Circuit 2	Circuit 2 as: Other
24	English	English locale id - en. Ll...
37	French	as: Other
41	Help	as: Other
44	Language	as: Other
47	Evaporator Leaving Water Temperature	Evaporator Leaving Wat...
50	None	Used on the DynaView f...
51	Off	as: Enumeration
53	Outdoor Air Temperature	Outdoor Air Temperature ...
55	Percent	as: Other
75	Unit	as: Other

Add Delete Copy Format Copy/Move Messages
Create Languages XML File

Message Context: MODBUS Base Loading Enable
Message Is: ☑ Devices ☑ Diagnostic Sources ☐ Diagnostic Targets ☑ Enumeration Detail ☐ LLID Index ☐ LLID
Message Included In: ☐ CTVConfigQuestions ☐ Configuration ☐ LCI_C Diagnostic ☐ Large OD ☐ Large OD Diagnostics ☐ Large OD Enumeration Values

Display Text
Language Filter: All Format Filter: All
Edit Language Edit Format

Format	Language	Display Text	Reviewed
Large OD 4...	English-Un...	MODBUS Base Loading Enal	☐
Large Based	English-Un...		☐

MODBUS Base Loading Enab
MODBUS Capacity Limit Input
MODBUS Chiller Mode Input (
MODBUS Diagnostic Encodin
Modbus Gateway Status Pack
Modbus Gateway Status Pack
MODBUS ICS Address
MODBUS ICS Baud Rate
MODBUS ICS Parity (1 == eve
MODBUS ICS Stop Bits (1 or 2
ModBus Interface Module
ModBus Network
MODBUS Remote Base Loadi
MODBUS Remote Chilled Wat
MODBUS Remote Chiller Run
MODBUS Remote Cond Pump
MODBUS Remote Condenser
MODBUS Remote Evap Pump
MODBUS Remote Evaporator
MODBUS Remote Hot Water
Modbus Run Command AFD 1
Modbus Run Command AFD 2
MODBUS Slave
Modbus Speed Command AFD
Modbus Speed Command AFD
Mode
Mode Component
Mode Transition Table Used C
Mode Transition Table Used C
Mode Transition Time Remaini

Add Delete

DataTrackerv2

File Tools

Product Selection: Symbio800_HTWR SW Part Number: 62000845

Variables Diagnostics Devices Translations

Translated / Not Translated: All
Display Language: English
Search: Apply Filter Clear Filter

Msg ID	Display Text
1	MODBUS Base Loading Enable
4	About
5	Active Chilled Water Setpoint
6	Active Current Limit Setpoint
17	
18	
24	
37	
41	
44	
47	
50	
51	
53	
55	
75	
81	
84	
104	
109	Open
113	Chilled Water Softload Target

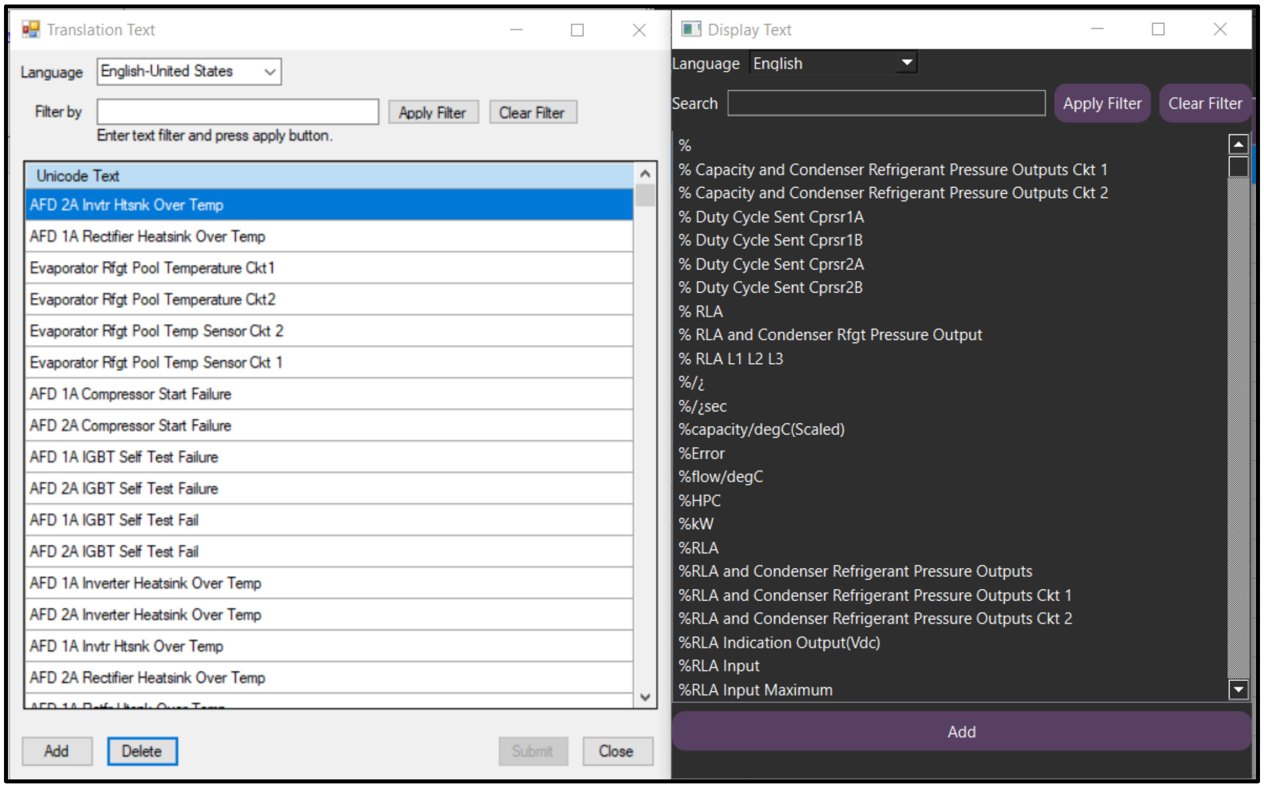
Modify Translated Text
Message ID: 1
Translated Text Format: Large OD 40 ASCII Char String
Translated Text Language: English
Display Text: MODBUS Base Loading Enable
Search: Apply Filter Clear Filter
Q_Model
Q_ControlFCODE
MODBUS Base Loading Enable
QDamperValveSize
Q_PrimaryInletSize
Q_OccupiedHeatingSetpoint
Q_OccupiedCoolingSetpoint
Q_UnoccupiedHeatingSetpoint
Q_UnoccupiedCoolingSetPoint
Q_MinCoolingFlow
Q_MinHeatingFlow
Q_CoolingMaxFlow
Q_MaxFlow
Submit Cancel

ID: 1 Display Text: MODBUS Base Loading Enable
Used in Current Project (Status)
☐ Device ☐ Diagnostic ☐ Enumeration Detail
☐ LLID Index ☐ LLID ☐ System
🟩 Unit ☐ Variable ☐ Other
Translate Message
☐ Do Not Translate 🟩 Translate
Display Text
Format: All Language: All

Format	Language	Display Text
Large OD 40 ASCII Char ...	English	MODBUS Base Loading Enable
Large Based	English	MODBUS Base Loading Enable

Add Edit Delete
Submit Cancel Create New Display Text

Figure 20: *DataTracker* Translated Text Selection Before and After



Translation Text

Language: English-United States
Filter by: Apply Filter Clear Filter
Enter text filter and press apply button.

Unicode Text
AFD 2A Invtr Htsnk Over Temp
AFD 1A Rectifier Heatsink Over Temp
Evaporator Rfgt Pool Temperature Ckt1
Evaporator Rfgt Pool Temperature Ckt2
Evaporator Rfgt Pool Temp Sensor Ckt 2
Evaporator Rfgt Pool Temp Sensor Ckt 1
AFD 1A Compressor Start Failure
AFD 2A Compressor Start Failure
AFD 1A IGBT Self Test Failure
AFD 2A IGBT Self Test Failure
AFD 1A IGBT Self Test Fail
AFD 2A IGBT Self Test Fail
AFD 1A Inverter Heatsink Over Temp
AFD 2A Inverter Heatsink Over Temp
AFD 1A Invtr Htsnk Over Temp
AFD 2A Rectifier Heatsink Over Temp

Add Delete Submit Close

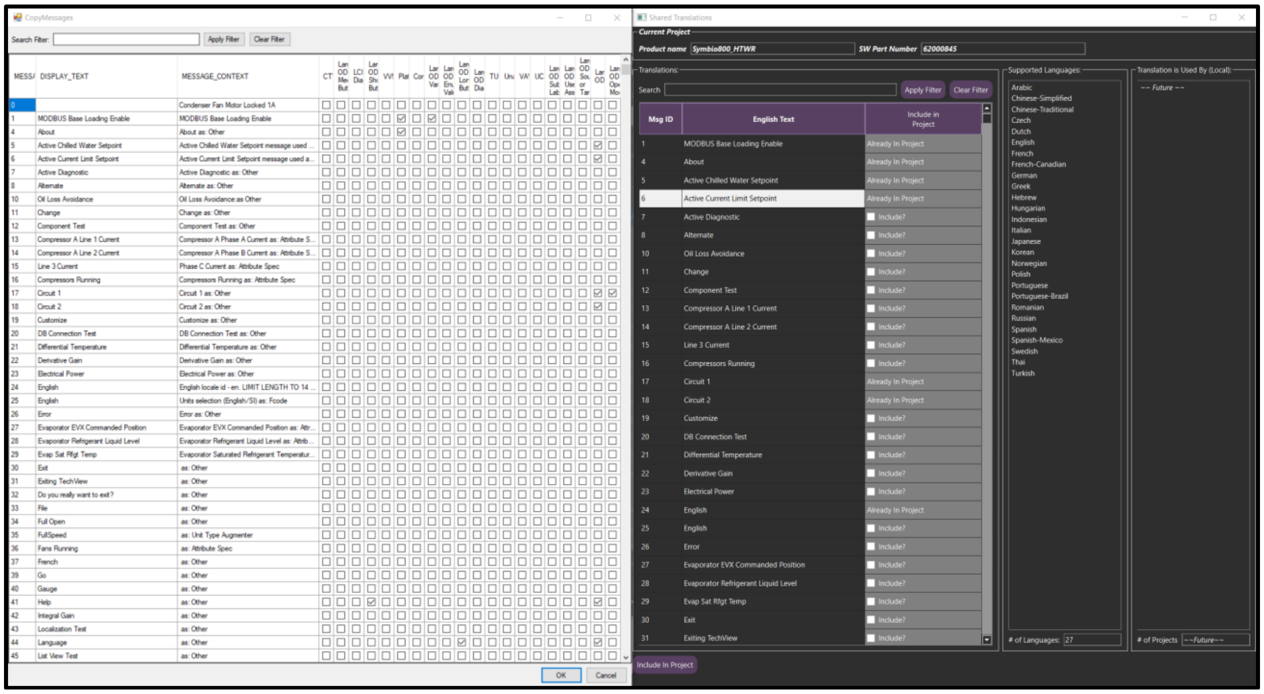
Display Text

Language: English
Search: Apply Filter Clear Filter

%
% Capacity and Condenser Refrigerant Pressure Outputs Ckt 1
% Capacity and Condenser Refrigerant Pressure Outputs Ckt 2
% Duty Cycle Sent Cprsr1A
% Duty Cycle Sent Cprsr1B
% Duty Cycle Sent Cprsr2A
% Duty Cycle Sent Cprsr2B
% RLA
% RLA and Condenser Rfgt Pressure Output
% RLA L1 L2 L3
%/¿
%/¿sec
%capacity/degC(Scaled)
%Error
%flow/degC
%HPC
%kW
%RLA
%RLA and Condenser Refrigerant Pressure Outputs
%RLA and Condenser Refrigerant Pressure Outputs Ckt 1
%RLA and Condenser Refrigerant Pressure Outputs Ckt 2
%RLA Indication Output(Vdc)
%RLA Input
%RLA Input Maximum

Add

Figure 21: *DataTracker* Display Texts Page Before and After


CopyMessages

Search Filter: [Apply Filter] [Clear Filter]

MESSA	DISPLAY_TEXT	MESSAGE_CONTEXT
0		Condenser Fan Motor Locked 1A
1	MODBUS Base Loading Enable	MODBUS Base Loading Enable
4	About	About as: Other
5	Active Chilled Water Setpoint	Active Chilled Water Setpoint message used ...
6	Active Current Limit Setpoint	Active Current Limit Setpoint message used a...
7	Active Diagnostic	Active Diagnostic as: Other
8	Alternate	Alternate as: Other
10	Oil Loss Avoidance	Oil Loss Avoidance:as Other
11	Change	Change as: Other
12	Component Test	Component Test as: Other
13	Compressor A Line 1 Current	Compressor A Phase A Current as: Attribute S...
14	Compressor A Line 2 Current	Compressor A Phase B Current as: Attribute S...
15	Line 3 Current	Phase C Current as: Attribute Spec
16	Compressors Running	Compressors Running as: Attribute Spec
17	Circuit 1	Circuit 1 as: Other
18	Circuit 2	Circuit 2 as: Other
19	Customize	Customize as: Other
20	DB Connection Test	DB Connection Test as: Other
21	Differential Temperature	Differential Temperature as: Other
22	Derivative Gain	Derivative Gain as: Other
23	Electrical Power	Electrical Power as: Other
24	English	English locale id - en. LIMIT LENGTH TO 14 ...
25	English	Units selection (English/SI) as: Fcode
26	Error	Error as: Other
27	Evaporator EVX Commanded Position	Evaporator EVX Commanded Position as: Attr...
28	Evaporator Refrigerant Liquid Level	Evaporator Refrigerant Liquid Level as: Attrib...
29	Evap Sat Rfgt Temp	Evaporator Saturated Refrigerant Temperatur...
30	Exit	as: Other
31	Exiting TechView	as: Other
32	Do you really want to exit?	as: Other
33	File	as: Other
34	Full Open	as: Other
35	FullSpeed	as: Unit Type Augmenter
36	Fans Running	as: Attribute Spec
37	French	as: Other
39	Go	as: Other
40	Gauge	as: Other
41	Help	as: Other
42	Integral Gain	as: Other
43	Localization Test	as: Other
44	Language	as: Other
45	List View Test	as: Other

[OK] [Cancel]

Shared Translations

Current Project

Product name: Symbio800_HTWR SW Part Number: 62000845

Translations:

Search [Apply Filter] [Clear Filter]

Msg ID	English Text	Include in Project
1	MODBUS Base Loading Enable	Already In Project
4	About	Already In Project
5	Active Chilled Water Setpoint	Already In Project
6	Active Current Limit Setpoint	Already In Project
7	Active Diagnostic	Include?
8	Alternate	Include?
10	Oil Loss Avoidance	Include?
11	Change	Include?
12	Component Test	Include?
13	Compressor A Line 1 Current	Include?
14	Compressor A Line 2 Current	Include?
15	Line 3 Current	Include?
16	Compressors Running	Include?
17	Circuit 1	Already In Project
18	Circuit 2	Already In Project
19	Customize	Include?
20	DB Connection Test	Include?
21	Differential Temperature	Include?
22	Derivative Gain	Include?
23	Electrical Power	Include?
24	English	Already In Project
25	English	Include?
26	Error	Include?
27	Evaporator EVX Commanded Position	Include?
28	Evaporator Refrigerant Liquid Level	Include?
29	Evap Sat Rfgt Temp	Include?
30	Exit	Include?
31	Exiting TechView	Include?

Supported Languages:

Arabic, Chinese-Simplified, Chinese-Traditional, Czech, Dutch, English, French, French-Canadian, German, Greek, Hebrew, Hungarian, Indonesian, Italian, Japanese, Korean, Norwegian, Polish, Portuguese, Portuguese-Brazil, Romanian, Russian, Spanish, Spanish-Mexico, Swedish, Thai, Turkish

# of Languages: 27

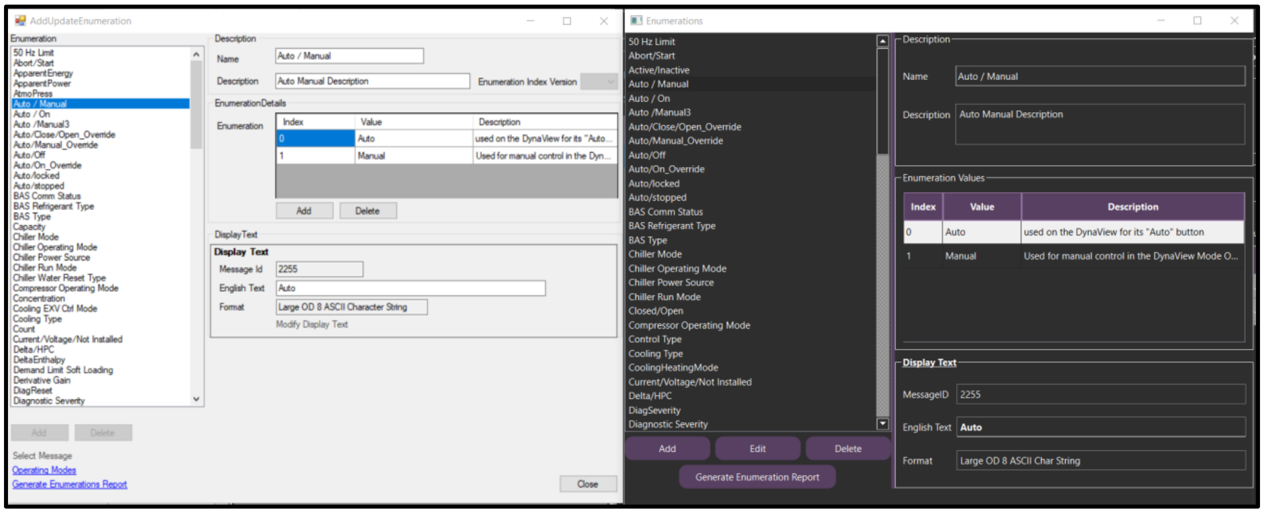
Translation is Used By (Local):

-- Future --

# of Projects: --Future--

[Include In Project]

Figure 22: *DataTracker* Shared Translations Page Before and After


AddUpdateEnumeration

Enumeration

50 Hz Limit, Abort/Start, ApparentEnergy, ApparentPower, AtmoPress, Auto / Manual, Auto / On, Auto /Manual3, Auto/Close/Open_Override, Auto/Manual_Override, Auto/Off, Auto/On_Override, Auto/locked, Auto/stopped, BAS Comm Status, BAS Refrigerant Type, BAS Type, Capacity, Chiller Mode, Chiller Operating Mode, Chiller Power Source, Chiller Run Mode, Chiller Water Reset Type, Compressor Operating Mode, Concentration, Cooling EXV Ctrl Mode, Cooling Type, Count, Current/Voltage/Not Installed, Delta/HPC, DeltaEnthalpy, Demand Limit Soft Loading, Derivative Gain, DiagReset, Diagnostic Severity

Description

Name: Auto / Manual

Description: Auto Manual Description Enumeration Index Version

EnumerationDetails

Enumeration

Index	Value	Description
0	Auto	used on the DynaView for its "Auto...
1	Manual	Used for manual control in the Dyn...

[Add] [Delete]

DisplayText

Display Text

Message Id: 2255

English Text: Auto

Format: Large OD 8 ASCII Character String

Modify Display Text

[Add] [Delete]

Select Message

Operating Modes

Generate Enumerations Report

[Close]

Enumerations

50 Hz Limit, Abort/Start, Active/Inactive, Auto / Manual, Auto / On, Auto /Manual3, Auto/Close/Open_Override, Auto/Manual_Override, Auto/Off, Auto/On_Override, Auto/locked, Auto/stopped, BAS Comm Status, BAS Refrigerant Type, BAS Type, Chiller Mode, Chiller Operating Mode, Chiller Power Source, Chiller Run Mode, Closed/Open, Compressor Operating Mode, Control Type, Cooling Type, CoolingHeatingMode, Current/Voltage/Not Installed, Delta/HPC, DiagSeverity, Diagnostic Severity

[Add] [Edit] [Delete]

[Generate Enumeration Report]

Description

Name: Auto / Manual

Description: Auto Manual Description

Enumeration Values

Index	Value	Description
0	Auto	used on the DynaView for its "Auto" button
1	Manual	Used for manual control in the DynaView Mode O...

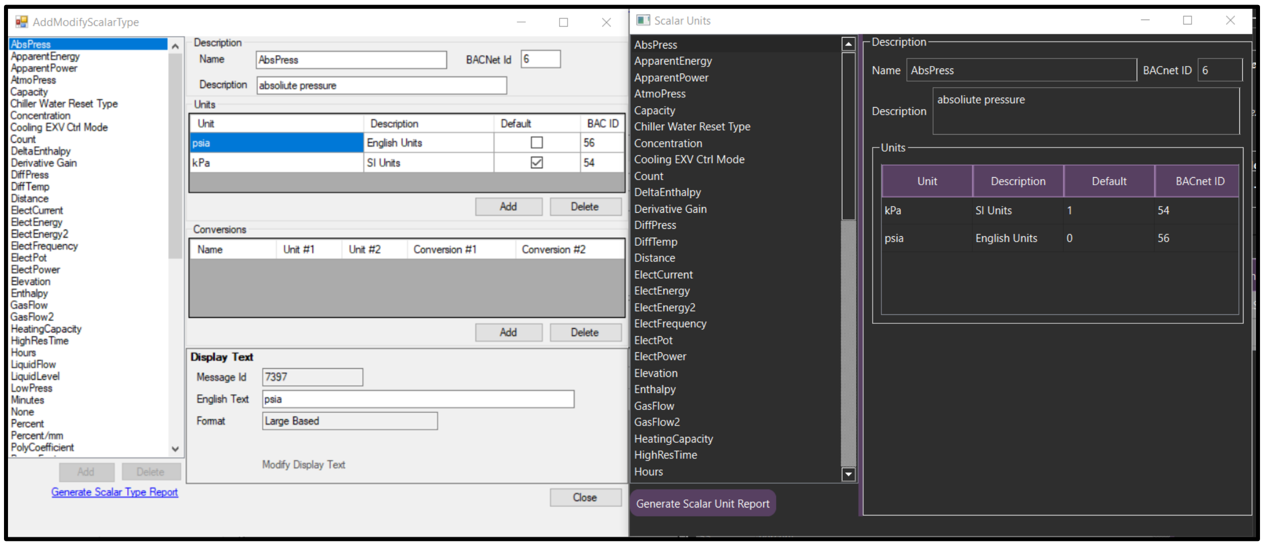
Display Text

MessageID: 2255

English Text: **Auto**

Format: Large OD 8 ASCII Char String

Figure 23: *DataTracker* Enumerations Page Before and After



Description

Name: AbsPress BACNet Id: 6

Description: absoliute pressure

Units

Unit	Description	Default	BAC ID
psia	English Units	☐	56
kPa	SI Units	☑	54

Conversions

Name	Unit #1	Unit #2	Conversion #1	Conversion #2

Display Text

Message Id: 7397

English Text: psia

Format: Large Based

Description

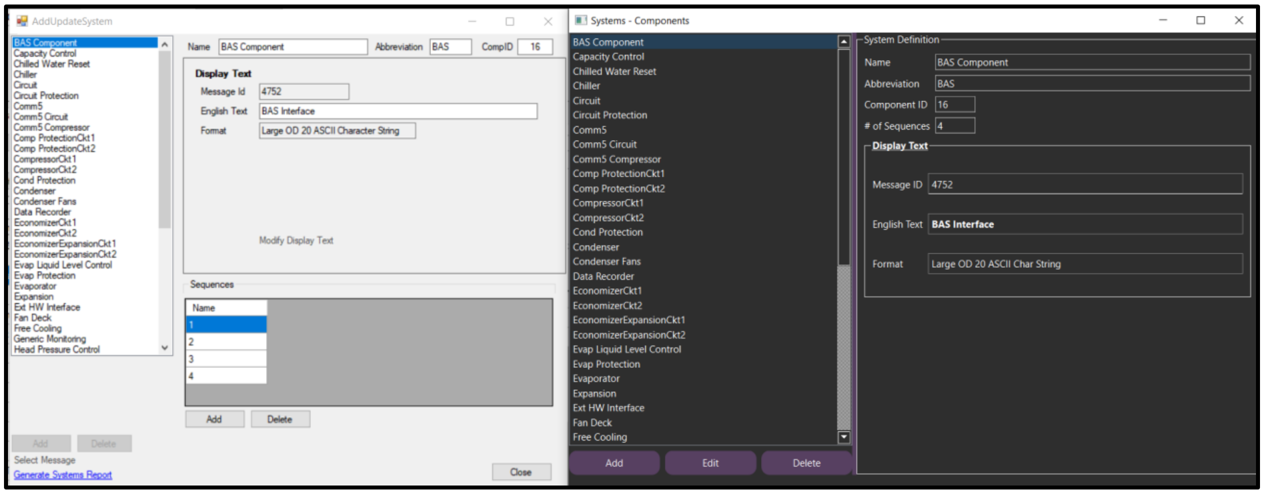
Name: AbsPress BACnet ID: 6

Description: absoliute pressure

Units

Unit	Description	Default	BACnet ID
kPa	SI Units	1	54
psia	English Units	0	56

Figure 24: *DataTracker* Scalar Units Page Before and After



Name: BAS Component Abbreviation: BAS CompID: 16

Display Text

Message Id: 4752

English Text: BAS Interface

Format: Large OD 20 ASCII Character String

Sequences

Name
1
2
3
4

System Definition

Name: BAS Component

Abbreviation: BAS

Component ID: 16

# of Sequences: 4

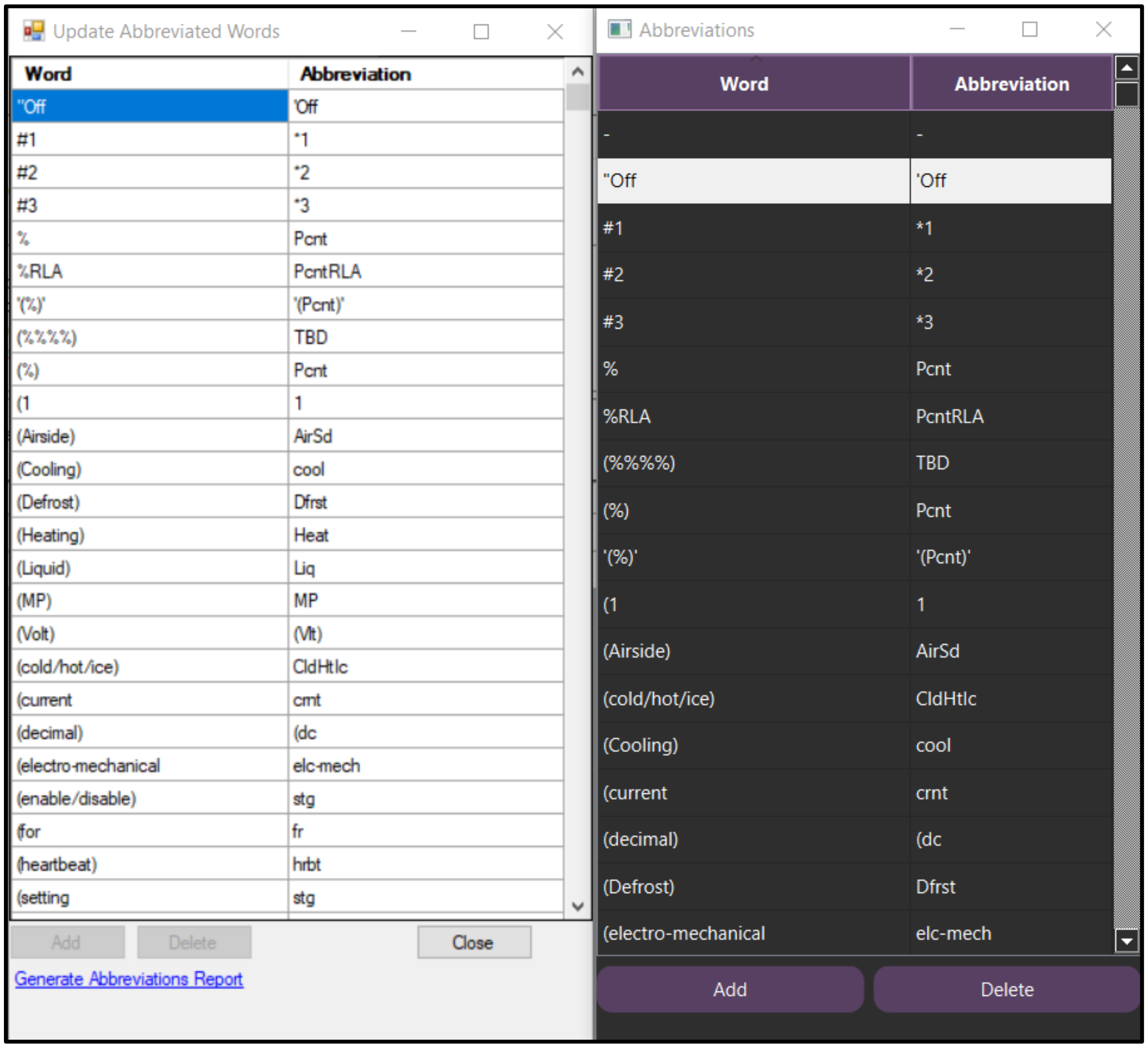
Display Text

Message ID: 4752

English Text: **BAS Interface**

Format: Large OD 20 ASCII Char String

Figure 25: *DataTracker* Systems-Components Page Before and After



Word	Abbreviation
"Off	'Off
#1	*1
#2	*2
#3	*3
%	Pcnt
%RLA	PcntRLA
'(%)'	'(Pcnt)'
(%%%%)	TBD
(%)	Pcnt
(1	1
(Airside)	AirSd
(Cooling)	cool
(Defrost)	Dfrst
(Heating)	Heat
(Liquid)	Liq
(MP)	MP
(Volt)	(Vlt)
(cold/hot/ice)	CldHtIc
(current	crnt
(decimal)	(dc
(electro-mechanical	elc-mech
(enable/disable)	stg
(for	fr
(heartbeat)	hrbt
(setting	stg

Word	Abbreviation
-	-
"Off	'Off
#1	*1
#2	*2
#3	*3
%	Pcnt
%RLA	PcntRLA
(%%%%)	TBD
(%)	Pcnt
'(%)'	'(Pcnt)'
(1	1
(Airside)	AirSd
(cold/hot/ice)	CldHtIc
(Cooling)	cool
(current	crnt
(decimal)	(dc
(Defrost)	Dfrst
(electro-mechanical	elc-mech

Figure 26: *DataTracker* Abbreviations Page Before and After