

FMH606 Master's Thesis 2024

Industrial IT and Automation - Industry Master

Development of configuration software for a test rig of electrical systems

Krister Pedersen

Course: FMH606 Master's Thesis, 2024

Title: Development of configuration software for a test rig of electrical systems

Number of pages: 104

Keywords: Software development, XML, Unified process

Student: Krister Pedersen

Supervisor: Nils-Olav Skeie, Co-supervisor: Leila Ben Saad

External partner: Ove Lehto Pedersen, Kongsberg Defence and Aerospace

Summary:

DSS Horten has an extensive test system designed to test and validate the products they design and manufacture for use in space. This test procedure is largely managed by an inhouse software called KNS-TS. KNS-TS relies on accurate XML files describing the configuration of hardware in any active test setup. These XML files are currently created and managed by hand. This thesis describes the work done in designing and implementing a software that can create these XML files. The focus of the thesis is on the complex relay matrices in use by anyone of these setups. The XML configuration files should be creatable from a clean sheet, or from import. The import can be from existing XML files or from design files with extension DXF. The software is developed using the unified process, an iterative, risk driven and use case based methodology. The software uses QT as a user interface library and is written in C++. The product must be developed with focus on future expansion of additional modules. The future goal of the software is to be able to create XML configuration files for several different modules. To achieve this the software must be well structured and ensure low coupling.

The software created during the thesis is shown to implement all the functional requirements given. The software has a structure that enables the development of future extensions.

Preface

I would like to thank Ove Letho Pedersen for the opportunity to work for you the last 3 years, and the continued support during my master's degree. I've gained valuable work experience while being allowed to primarily focus on studies.

I would also like to thank my supervisors, Nils-Olav Skeie and Leila Ben Saad, for their interest in my thesis. I appreciate the valuable contributions you've brought me weekly during this project, and for the helpful feedback on my thesis.

Porsgrunn, 14.05.2024

Krister Pedersen

Contents

1	Introduction	6
1.1	Background	6
1.2	Objective	6
1.3	Contribution	7
1.4	Report structure	7
2	System description	8
2.1	System environment	8
2.2	System input	10
2.3	System output	10
2.4	System development	10
3	Literature review	12
3.1	XML history	12
3.2	XML format	12
3.3	XML usage	14
3.4	XML in DSS	14
4	Analysis, design, and implementation	15
4.1	System requirements	15
4.2	Use case analysis	16
4.3	Iterations	19
4.3.1	<i>Iteration 1</i>	19
4.3.2	<i>Iteration 2</i>	22
4.3.3	<i>Iteration 3</i>	23
4.3.4	<i>Iteration 4</i>	24
4.4	System architecture	26
4.5	Implementation	31
5	Results	36
5.1	Main window	36
5.2	Manual input	37
5.3	Export XML	39
5.4	Import XML files	43
5.5	Import dxf files	45
5.6	Testing	48
6	Future development	50
6.1	Implementing epc configuration	50
7	Discussion	52
8	Conclusion	53
8.1	Possible improvements	53

Nomenclature

DSS - Division for space and Surveillance. The customer of the project.

DUT - Device under testing. Common word used for describing the product being tested.

EPC - Electronic power control. A special type of jig used in the test system at DSS.

Grasp - General Responsibility Assignment Software Patterns.

(G)UI - (Graphical) user interface

HTML - HyperText Markup Language. Format used for visual presentation.

KDA - Kongsberg Defence and Aerospace. The customer company.

KNS-TS - Kongsberg NorSpace Test System. The automation software used for testing at DSS.

QT - Graphical user interface library used in the project.

SSD - System sequence diagram. A diagram showing the sequence of interactions between users and the system.

XML - Extensible markup Language. A file format used for representation of objects.

1 Introduction

This chapter will give an introduction to the work done in the master thesis. It will give background for the work and describe the objective of the thesis. It will also briefly discuss the contribution of this work compared to previous work. Lastly it will give an overview of the entire report structure. Working methods and scope will be further explained in later chapters.

1.1 Background

Kongsberg Defence and Aerospace (KDA) is a Norwegian tech company, focused on the defense and space industry. This thesis describes work done at KDA's Division for Space and Surveillance (DSS), located in Horten, Norway. DSS produces electrical equipment, like filters, receivers, and transmitters for use in space.

Space equipment has high requirements regarding quality assurance. The inability to repair or replace broken equipment after launch makes any error extremely expensive. To make sure the equipment made at DSS Horten does not fail in space, every product is rigorously tested before being shipped to its customers. These tests are done in different environments. Temperature changes, vacuum chambers and electromagnetic compatibility are some of the main environmental changes every product goes through.

To make the testing procedure as reliable and effective as possible, DSS Horten has developed its own testing software. This software is named KNS-TS. KNS-TS automates major parts of the testing process. To be able to do this automation KNS-TS uses several XML files. Configuration files with information about testing procedure, instruments and wiring are all required for KNS-TS to perform tests.

Currently, the creation of these XML files is done by hand. These configuration files are unique for every setup, usually multiple setups for each product. Manual edits are required for any customization. This is a difficult and time-consuming process. Manual work also means human error, which leads to additional time consumption identifying and correcting these errors.

1.2 Objective

DSS Horten would like to simplify the creation and maintenance of these configuration files. This project will be working on one of these XML files, the one describing jigs. This file contains information of different premade jigs at DSS Horten. They all have different tasks in the system. The main work in this thesis is focused on the relay matrix jigs. The jigs consist of a set of different relays of varying types and the wiring in between these. Exactly how this wiring is setup, which relays are included and how they are connected to the testing computer need to be described in the configuration file.

The objective of the project is to develop a software to automatically create XML files for configuration of jig in KNS-TS. The requirements given by the company will be described in chapter 4.1.

The task description can be found in Appendix A.

1.3 Contribution

The software developed for this project fills a unique gap in the current environment at DSS Horten. The company is missing a solution for creating and maintaining the configuration XML files needed to configure the inhouse software currently in use. The current manual solution is ineffective and error prone. Creating a software to cover this task would be very beneficial as a long-term solution. The software is considered a starting block for further development, and with further development it should become the long-term solution for configuring this part of the XML configuration for KNS-TS.

1.4 Report structure

Chapter 2 contains system description. The environment the new software is created for, the input and output of the system, and the development process is described in this chapter.

Chapter 3 contains a literature review of XML. It briefly touches on the history of the file format and the uses it has today. It then describes which role XML plays at DSS today, and why it is important for the software being created.

Chapter 4 contains analysis, design, and implementation of the current system. It describes the system requirements analysis and its results. It goes through the different use cases and the iterations of development the system went through. Lastly it describes the architecture of the software, and some implementation details.

Chapter 5 contains results of the development. It goes through the different functionalities of the software, and the testing performed on the system.

Chapter 6 contains futered development, both planned now and possible further extensions.

Chapter 7 contains discussion of the thesis.

Chapter 8 contains the conclusion of the report.

2 System description

This chapter contains a description of the software being created. It describes the system, its surrounding environment, the different files it reads and writes, and the development process.

2.1 System environment

The products manufactured at DSS Horten are meant for use in space. These products can be transmitters, receivers, or filters, amongst others. Any malfunction in space products is very expensive. Both in cost for the customer and for the reputation of DSS. To avoid any malfunction, testing is of the utmost importance. This is why DSS Horten have made its own test software.

The test system in use at DSS Horten is called KNS-TS. This software automates a large part of the rigorous testing process at the company. Depending on the device being tested, this could require a large set of instruments, wiring and relays. KNS-TS must have full control of all these parts, in addition to other information like test procedures and test environments. To feed all this configuration data to KNS-TS, a set of XML files are used. These XML files have been manually updated for every device, to reflect the environment it is being tested in.

This is where the configuration software fits in. To lighten the load on manual edits of XML files and enable imports from files made in design. This would lighten the load on certain resources at the company and allow for more effective and user-friendly upkeep of these configurations.

The configuration software, with initial focus on relay matrix configuration in this project scope, should create valid XML files on the format currently in use by KNS-TS. The software should be able to read and create XML files. Additionally, it should be able to read DXF files created from design tools currently in use at the company. It has to convert relay information in these files to XML format, while allowing user edit.

2 System description

Figure 2-1 shows the existing test system, with the new software marked in green. In today's system the user is manually editing the XML file.

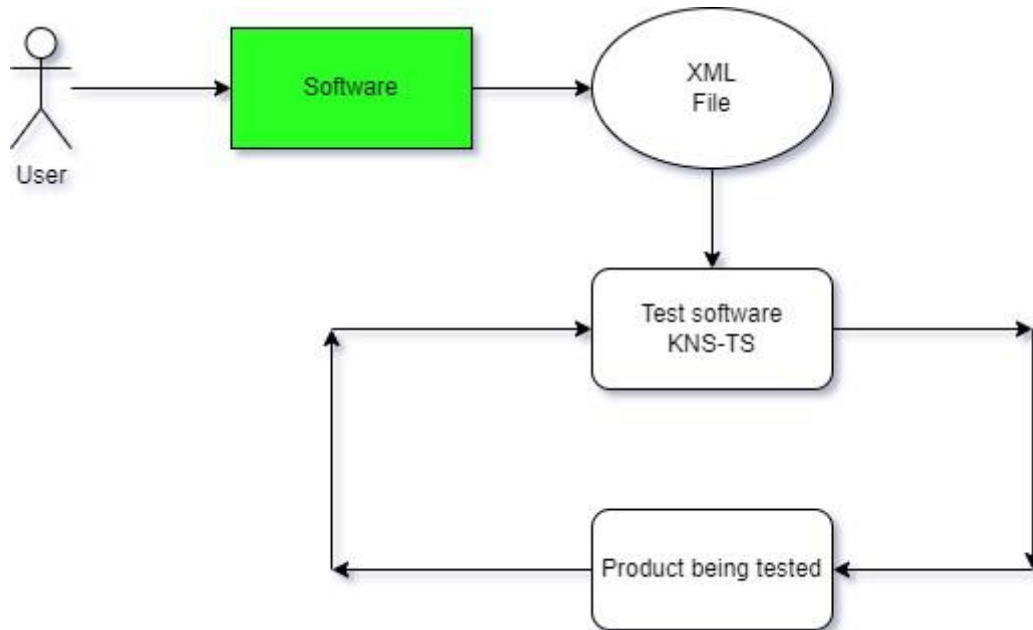


Figure 2-1 The full test system.

A life cycle of a specific configuration can vary widely. If the test results are as expected, the configuration can be active for only minutes. Other configurations can last for weeks during debug. In any case the new software should be helpful for early error detection and quick alteration of configurations.

The relay matrices being configured can be relatively complex. They can contain 30+ relays, with wiring to instruments, devices being tested and in between each other. This creates a vast matrix with a lot of possible routings. The system should have information of both the relays in the matrix and be able to save different routings. These routings will consist of a set of relays, and a position for each of these. Allowing for easy switching on multiple relays with only one action.

The physical relays are of different types. Each type has different qualities, like number of positions or operating voltages. This information is also needed for KNS-TS to have full control over testing. This data must be included in the configuration XML sheet.

The current test software, KNS-TS, is a massive ongoing software project. The main coding language is C++, and the company wishes to keep any new software in the same language. The software department in the company is rather small, and keeping code in the same language ensures that the company have resources available with experience in the language of every software.

With the same reasoning, QT was chosen as a user interface library. QT allows for a interactive user interface across multiple languages, with good support for C++.

2.2 System input

The test system uses XML files as both input and output. XML files are used to tell the test software, KNS-TS, how the hardware is connected. Instruments, wiring, devices to be tested, and test procedure must all be configured correctly for KNS-TS to be able to start the test procedure.

The input files being worked on in this new software are XML files and DXF files. Both these files should be read, parsed and visualized in the new software. The XML files are manually made and edited today.

The DXF files are automatically generated from sketches of the entire test system. These sketches are done by the design team at the company. They are made using a tool called Altium Designer. Extracting data from these DXF files can be challenging, as they can grow very big. The example file provided for development purposes consists of 170 000 lines. To be able to extract data the company has decided on a new naming convention. This convention makes sure the name of the relay is connected to the type of relay, and the hardware connection to the test pc.

The new configuration software will also use an XML file containing information about available relay types as input. This XML files will have data on type name and relay attributes.

2.3 System output

To store system output, a PostgreSQL database structure is used. Every test performed is pushed onto a common database. From this database the results can be checked against the customer specifications, and large test reports can be generated. These reports are sometimes exported as XML, by customer wishes.

The new configuration software will also have XML output. It will not use a database, but store XML files at the path specified by the user. The format of these files will be described in the results chapter, 5.3.

The XML file format created by the software will be based on working versions of the configuration files. The XML files are of version 1.0 using utf-8 encoding.

2.4 System development

The development process is based on the unified process [1, p22]. It is an iterative development process. The development will use iteration cycles of 2 weeks. The process uses use cases as the primary modeling tool. Every iteration cycle is based around 1 use case. After the development of the use case is completed, the software goes through a set of tests, customer feedback, and a new use case is selected for the next iteration.

To reduce the risk in the development process, the use cases are sorted from most important to least important. The most important use case is selected for the first iteration. This helps ensure that the most important functionality of the software is optimized.

2 System description

The start of the process consists of system analysis. The main focus is on collecting requirements and converting the requirements to use cases. The requirement analysis is done using FURPS+. The concept and the results of it is described in chapter 4.1. From the analysis a use case diagram will be developed. This diagram will be used to discuss the process with the customer. The use case diagram and the iterations will be described in chapter 4.2 and 4.3.

The unified process has 4 main phases [1, p23]. The inception phase is the start of the project. Cost estimations, decide if the project is worth running and time management are important choices made in this phase.

The next phase is the elaboration phase. This phase is where the requirement specification and the identification of use cases are done. Choices regarding design and architecture are made in this phase. Startup of development is also included.

The main phase is called the construction phase. The first use case is selected and work on development is intensified. More iterations of analysis, design and implementation of use cases are included. Most of the iterations of development should be in this phase.

The last phase is called the Transition phase. This is where the software is transitioned into a mature version of the software. Ready for deployment.

Version control is used to ensure the work is not lost by technical failure. DSS Horten uses SVN for version control. The company has reserved a space for the software to be committed to. The full source code is not made open for everyone. The report will show the functionalities of the system and show small examples of implementation details.

3 Literature review

The software being developed relies heavily on the use of the XML format, both reading and writing XML files. With this reasoning a literature review of the XML was performed. This chapter will give a short introduction to the XML format. It will mention the most important historical events that made XML what it is today. It will also describe how it is used, across the world and in DSS Horten specifically.

3.1 XML history

Extensible markup Language (XML) is derived from Standard Generalized Mark-up Language (SGML) [2]. SGML was invented in 1986, and is still used as a standard today [3]. The most popular SGML standard today is the HyperText Markup Language, known as HTML. HTML is used in every web browser today. Its syntax is layout oriented, making it ideal for visual presentation. HTML however does have some limitations.

The limitations of HTML lead the World Wide Web Consortium (W3C) to start the work on an alternative. This work was led Jon Bosak. He was an engineer with extensive SGML experience from managing technical documents from big companies like Sun Microsystems in the 1990s and 2000s. This alternative was published under the name XML in 1998.

HTML was meant for transmitting visual data, while XML would focus on transmitting generic data structures. A big limitation of HTML was that it had a smaller standard set of tags available. These tags were used to style the contents and make it into a graphic web site. XML would let the user specify their own tags. This meant that every user could adapt it to their dataset, making it suitable for generic information exchange.

XML version 1.0 was originally defined in 1998 but has since then been updated. The last update to version 1.0 was done in 2008, this was the 5th edition. This is the most commonly used version of XML and is still recommended for general use [4].

XML version 1.1 was published 2004, with its latest edition, the 2nd edition, being published in 2006. This version enables the use of scripts and characters not available in Unicode 3.2, as well as some line-ending characters in use by IBM technology. It has very little use today [5].

3.2 XML format

XML is a markup language with defined rules for data presentation. These rules make the files easy to read and interpret for the receiver. Making the data transfer effective. This does however mean that any errors from the rules will invalidate the XML file.

XML files are called documents. The tag `<xml>` marks the start of every XML document. This is what the software will look for before it starts reading the file. The end tag `</xml>` marks the end of the file.

The XML file begins with a declaration. This contains some standard information needed to read the file. This is usually XML version and encoding format. An example of a declaration is: `<?xml version="1.0" encoding="UTF-8"?>`.

3 Literature review

The main content of the file is called elements. Every element can contain text, attributes or child elements. Consider this example XML file:

```
<Relay matrix>  
  <Relay Positions = "4">  
    <Name>K01</Name>  
    <Type> 87104b </Type>  
  </Relay>  
</Relay matrix>
```

The start element of every file is known as the root element. In this small example that would be Relay matrix. A child element to the root element is Relay. Relay has an attribute called position specified in the start tag. In this example the value of positions for this relay is 4. The child elements of relay are name and type.

These elements have text in between the start and end tag. This text is called XML content, or data. Information in between tags is known as data, while the tags and attributes are metadata.

An XML document that follows all the rules is known as a well-formed document. Additionally, schemas can be used to make sure the document is valid. The schema makes specific rules and has a list of valid attributes and tag names in the document.

3.3 XML usage

XML is used for serialization of data, meaning the conversion to and from binary data. The strict rules of the format allow for effective serialization. The format makes for effective reading for both humans and machines.

It is used for data transfer in millions of applications today. Everything from small private application to massive applications like Microsoft Office, used worldwide. It has a massive usage in web applications. It is used among other for representing[6]:

- Technical documentation
- Configuration data for software
- Book data
- Transaction data
- Invoice data

It allows for structured information to be passed from computer to computer, or from computer to people.

XML is extensively used in data transfer applications. Data is often transferred over the internet, intended for many different types of receivers. It could be web applications, desktop applications, mobile applications, or databases[7]. By using an effective standard format like XML, the data can be interpreted by all the different receivers. The receiver only has to know the format of the data to be able to use it as it should.

Configuration data for software will be the main XML usage in the software created in this project. The XML format allows for the customization of objects necessary to describe the hardware configuration for the active test setup.

3.4 XML in DSS

XML is widely used in DSS Horten. The KNS-TS software both reads and writes XML documents. The test software generates reports for customers in XML. Several XML files are used as input to describe the hardware and the test procedure of each individual device being tested. The current process for XML configuration can be seen in Figure 3-1. The project will eliminate the need of an expert and make the upkeep more accessible to other resources at DSS.

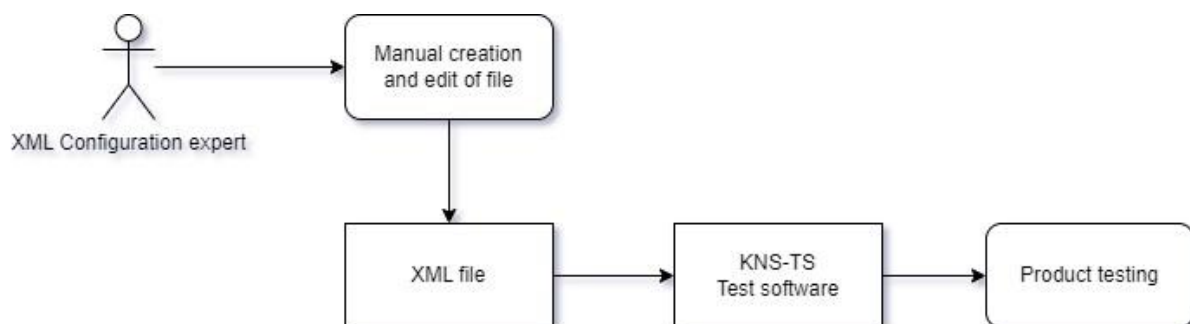


Figure 3-1 XML configuration in current system.

4 Analysis, design, and implementation

This chapter describes the analysis, design, and implementation of the software.

Communication with the customer, DSS Horten, gave an overview of the requirements of the software that were to be developed. These requirements were analyzed using FURPS+, as seen in. From this document a use case analysis was completed. Each use case was given 1 iteration of development. These are described in 4.3.

The system architecture is described in 4.4. Lastly some implementation details are described in chapter 4.5.

4.1 System requirements

System requirements were given by the customer at project startup. These were discussed in a formal meeting shortly after. Based on these requirements a FURPS+ analysis was completed. The full analysis can be found in Appendix B.

A FURPS+ analysis is a method for collecting requirements. Each letter stands for a different type of requirement [1, p66].

Functional requirements show the main functions of the system. These are described as use cases later in the analysis.

Usability describes user and software interaction. This software uses QT as a GUI tool. Additionally, XML and DXF is used as input files, and XML is produced as output from the system.

Reliability requirements for the system. In this software this describes error handling and validation of output XML.

Performance requirements. There are no hard performance requirements on this system. A performance requirement was still made to avoid excessive time spent on import functionality.

Supportability requirements for further development and long-time usage. The software describes in this thesis should be a starting point for further development. More modules should be added later, and the created software should support easy development of these extensions.

+ Design challenges and limitations of the system. These describe technology choices made by the customer in this software.

4.2 Use case analysis

The functional requirements found in 4.1 was used as a basis for a use case analysis. The 5 functional requirements were found to be suitable for 4 use cases. The external systems for the software were identified as the user, XML file, and DXF file. The full diagram can be found in Figure 4-1. The use cases are sorted top to bottom from most important to least important functionality. This was discussed with the customer, DSS Horten. Each one of these use cases would be covered by a separate iteration in the development cycle.

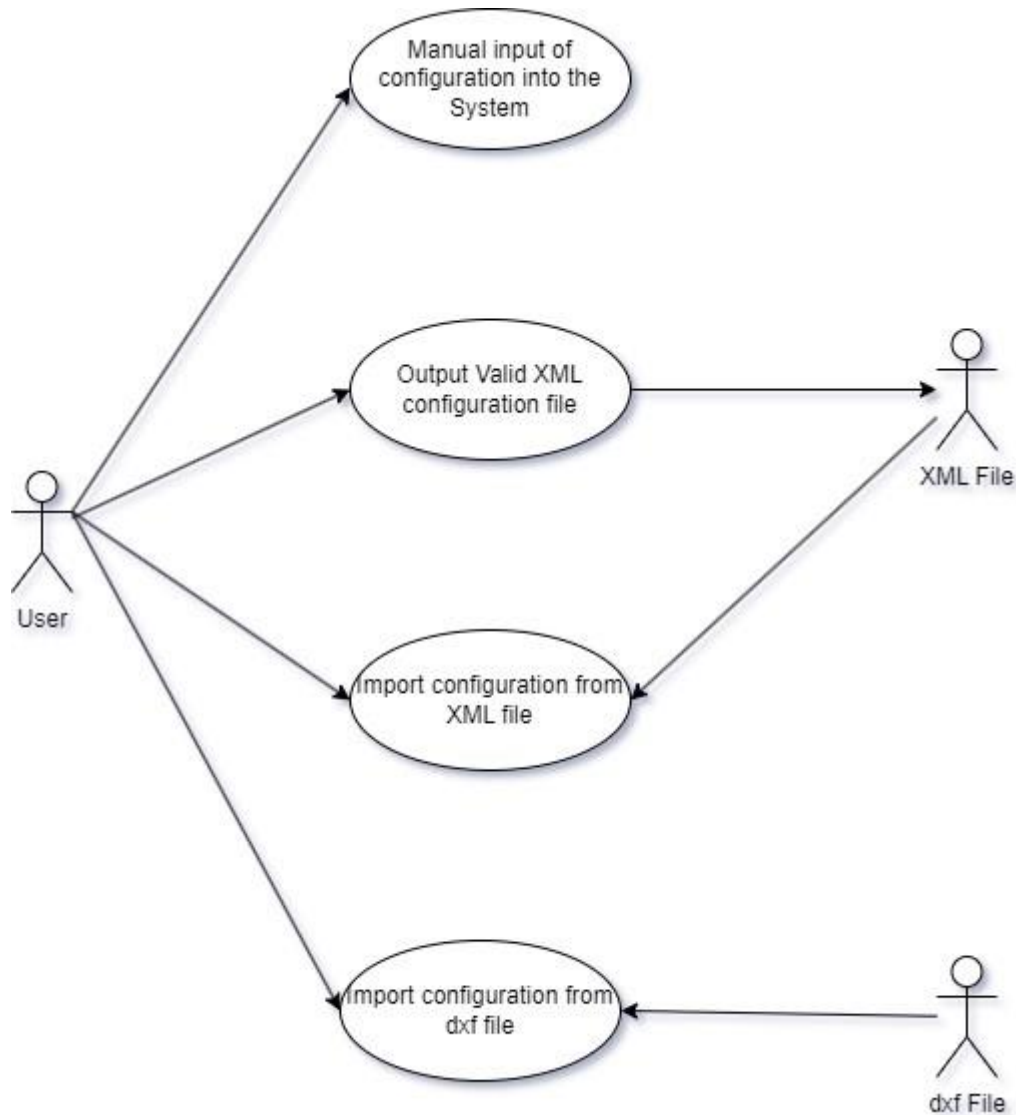


Figure 4-1 Use case diagram.

The first and most important use case is the manual input of configuration into the system. This use case would let the user create a configuration. This would be done by using GUI elements like buttons, combo boxes, and text fields. This would encompass adding new relays, updating a relay with changed data, and deleting a relay from the configuration. The same functionality should be possible for routings.

4 Analysis, design, and implementation

The second use case is the output of valid XML data in a configuration file. This would require a conversion of the data stored in the system, created by the first use case, into XML data. This XML data would have to be on the format currently used by the other software at DSS Horten, KNS-TS. Error handling would also be required to ensure that the successful or unsuccessful writing to XML was reported back to the user.

The third use case is the import of XML configuration files. The customer wanted the possibility of re-using existing XML files. By allowing the software to import these and converting them to the data format used in the software, the user would be able to edit the configuration using the new software. This would allow for error-correction and the reuse of configurations for almost identical setups.

The last use case is the import of design files, DXF files, into the software. This would allow the user to take files created in the design process of the test system and convert them into the data format used in the software. These files contain the information needed to identify the relays needed in the configuration output. It does however not contain any information about routings. These would have to be added manually by the user, as covered in use case 1.

In addition to the use case analysis, a couple of user interface sketches was created. These were shown to the customer to confirm a common understanding of the software that was to be developed. These sketches can be seen in Figure 4-2 and Figure 4-3.

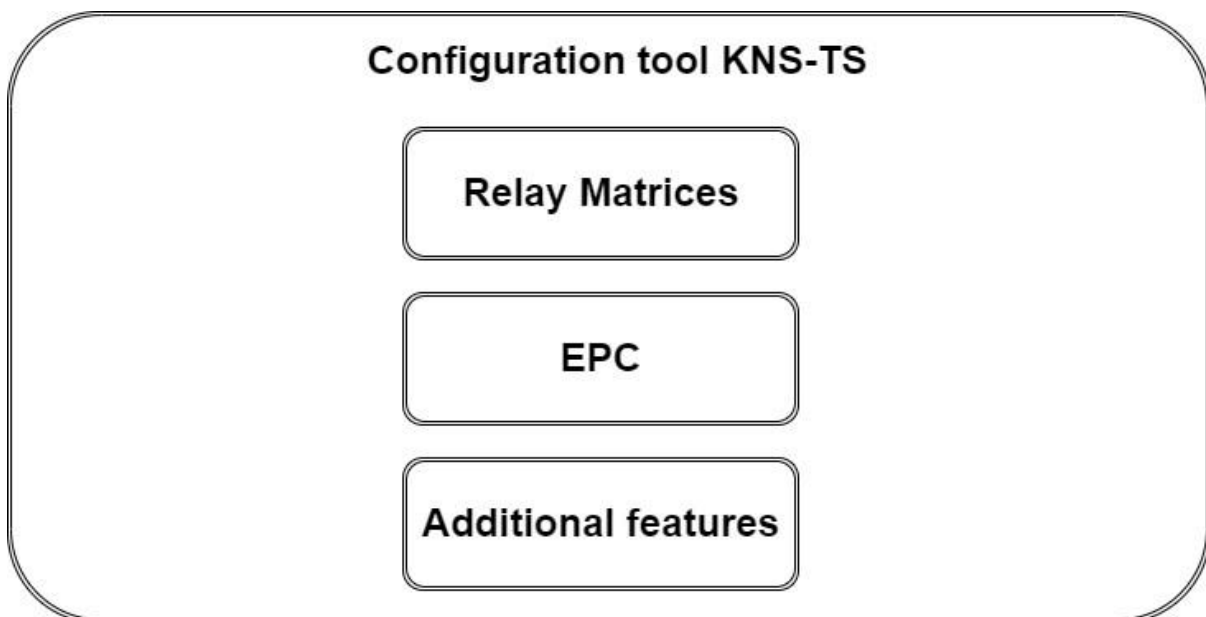


Figure 4-2 Sketch main menu.

4 Analysis, design, and implementation

Relay Matrix Config

Path:

Name:

Relay Type:

Dut Position:

Module:

Bank:

Relay(SW):

Routing name:

Dut Position:

Name:

Relay:

Position:

Relay(s):

Path:

Relays						
Variables	Name	Type	Dut pos	Module	Bank	Relay(SW)
Relay 1	K01	AG_87204_40GHZ_4W	2	1	3	2
Relay 2	K02	AG_87204_40GHZ_6W	1	3	2	1
Relay 3						

Routings			
Variables	Routing Name	Dut Pos	Relays(s)
Routing 1	DUT Input	1	K01_1, K05_5, K15_2
Routing 2	DUT Input	2	K01_2, K02_1, K09_1
Routing 3			

Figure 4-3 Sketch Relay matrix window.

The main menu is a simple window with buttons linked to new windows. In this development cycle, the window will only consist of a relay matrix and an EPC button. The EPC button will not be linked to anything but is meant as an entry point for the EPC window in later development. Closing this window should close the system.

The relay matrix window has more complex design. The data currently stored in the software is displayed by 2 tables on the right side of the window. Each of these tables have a set of text fields, combo boxes and buttons connected to them. These elements should be used to manipulate the data contained in the tables.

The import functionality is placed at the top of the sketch. It consists of a simple path field and a button to start the action. Export has the same functionality at the bottom of the sketch.

4.3 Iterations

The iterations will be described in this chapter. Each iteration describes the work done for 1 separate use case.

4.3.1 Iteration 1

The first iteration of development was used to create a system for manual input of data. The goal was to create an application with a graphical user interface that could collect, manipulate, and show the necessary data to create a configuration file.

The iteration started with an analysis of the use case. This analysis resulted 3 documents. Firstly, a Fully Dressed Use Case Diagram, a FDUCD, describing the use case and showing the different possible scenarios in the use case. Section 8 and 9 of the FDUCD can be seen below. The full FDUCD for every use case can be found in Appendix C. These sections describe the main success scenario and extensions. The main success scenario is the optimal order of operations in the use case. For this use case this is the add button. Any alternative action should be covered by the extensions. In this case that consists of the 2 other buttons, update and delete, as well as error when checking input values.

Main success scenario:

1. Get combo box data from XML file.
2. Add data using text fields and combo boxes.
3. Press “add” button.
4. Confirm valid data.
5. Add data to table.
6. Update GUI with new data.

Extensions:

Update button:

- 2a Select data from table.
- 2b Data for selected data shown in GUI.
- 2c Change 1 or more fields in GUI.
- 2d Press “update” button.
- 2e Confirm valid data.
- 2f Go to step 6.

Delete button:

- 2a Select data from table.
- 2b Data for selected data shown in GUI.
- 2c Press “delete” button.
- 2d Remove data from table.
- 2e Go to step 6.

4 Analysis, design, and implementation

Data checks:

4a, 11a Invalid data. (Missing data, invalid values, duplicate data)

4b, 11b Show error message.

4c Go to step 2.

A system sequence diagram, SSD, was then created using the scenarios from the FDUCD. This diagram shows the interaction between the users and the system. The SSD for iteration 1 can be seen in Appendix D. The diagram shows the flow of the program in different loops. It's separated into loops for every extension in the FDUCD.

The first iteration also required the implementation of the user interface. An implementation of the sketches discussed in 4.2 was done. The interface can be seen in Figure 4-4. The stylesheet of the user interface is taken from existing software, KNS-TS. A few alterations was made to the stylesheet to improve the visual presentation of the new software.

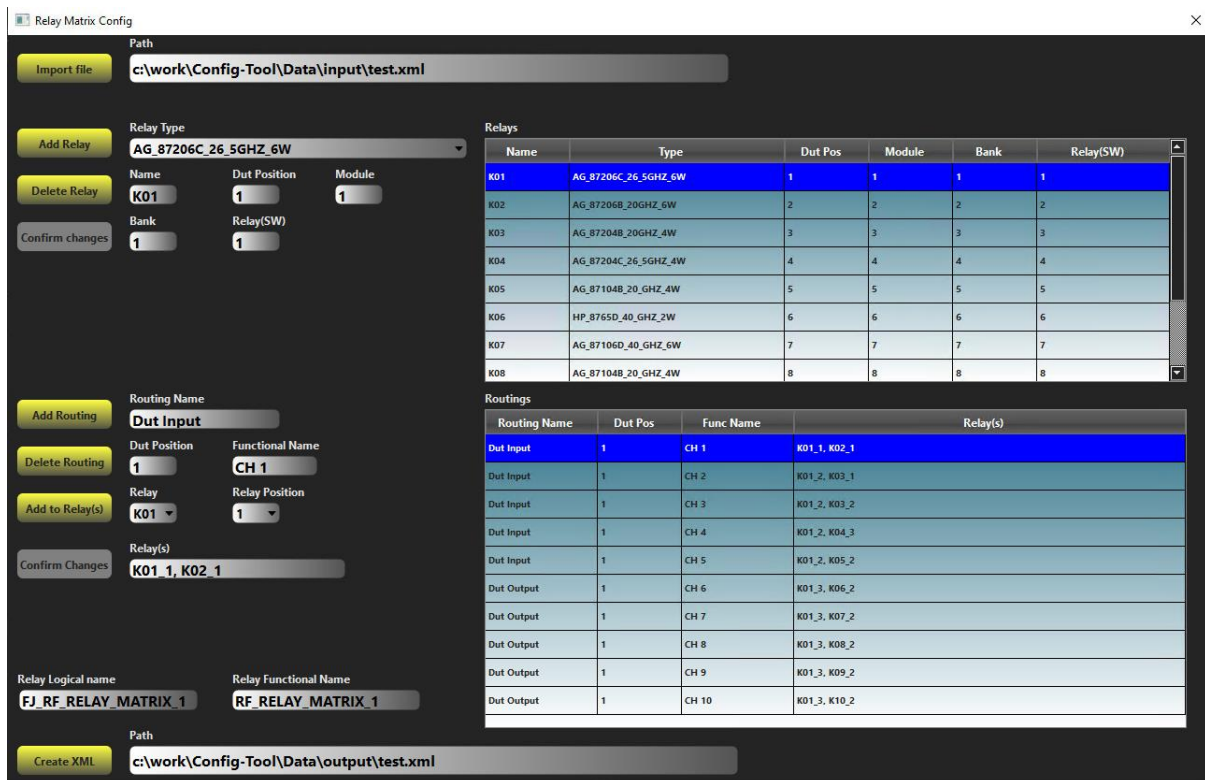


Figure 4-4 Gui implementation.

The 3rd document created was a sequence diagram. This diagram shows the interactions between different parts of the system. To do this, classes had to be identified, and their responsibilities had to be decided. The sequence diagram for iteration 1 can be seen in Figure 4-5.

4 Analysis, design, and implementation

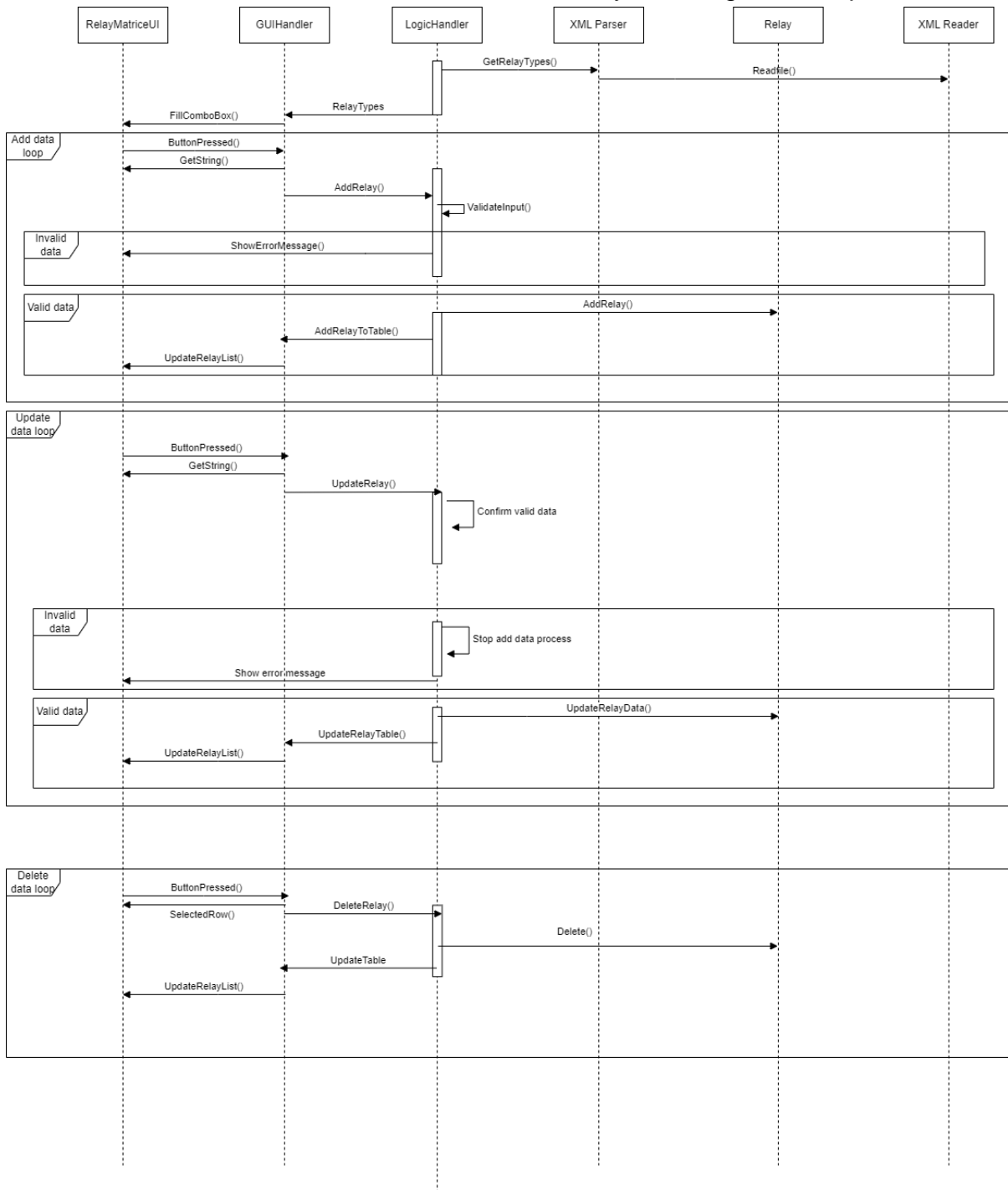


Figure 4-5 Sequence diagram for iteration 1.

To identify classes and responsibilities, several design patterns were considered [1, p192]. It was decided to implement a solution based on the façade pattern. This pattern describes a single point of contact for a subsystem. In the sequence diagram these contact points are called handlers. The UI layer is considered a subsystem, and it has the GUIHandler as the only point of contact. This means that any communication with other layers is done through this class. This is done to create a good structure for the software and make the software easier to refactor later. By using the façade pattern, it should be possible to swap out the

4 Analysis, design, and implementation

entire UI layer without making any changes to the logic layer or the data layer. The same should be the case for any layer.

These handlers will also follow the singleton pattern. This pattern tells us how to implement a static function in the class that ensures that only a single object of the class will be allowed to be created.

Further use of design patterns will be described in chapter 4.5.

4.3.2 Iteration 2

Iteration 2 started as iteration 1 by defining a FDUCD. Section 8 and 9 can be seen below, and the full document can be found in appendix C. The SSD created from these sections can be found in appendix D.

Main success scenario:

1. Data imported/input into system.
2. Add valid path into the output path field.
3. Press the Export button.
4. Convert relay and routing data to XML data.
5. Create the XML file at chosen path.
6. Confirm if export was successful.
7. Tell user of successful/unsuccessful export.

Extensions:

3a Invalid path.

3b Tell user of invalid path.

3c Go to step 2.

4a Invalid data.

4b Tell user of invalid data.

4c Go to step 1.

The main success scenario describes the user filling the system with data, and starting the export action in the system. The system then converts and formats the data to XML and writes the XML data to file. The status of the write operation is reported back to the user.

The extensions describe the scenarios of invalid path and invalid export data.

Sequence diagram for iteration 2 can be seen in Figure 4-6. The figure shows the different classes involved in the export operation. The button press is handled in RelayMatriceUI and sent to the GUIHandler. This class is responsible for communication with the logic layer, via class LogicHandler. LogicHandler gets data from the Relay and Routing classes and sends this data to the dataHandler. This class activates XMLWriter. The result from XMLWriter is then processed back to RelayMatricUI. This process ensure that all interlayer communication

4 Analysis, design, and implementation

is handled by handlers. This allows the software to manage dependencies efficiently. An example of the XML file generated by the software can be found in the result part of the report, chapter 5.3.

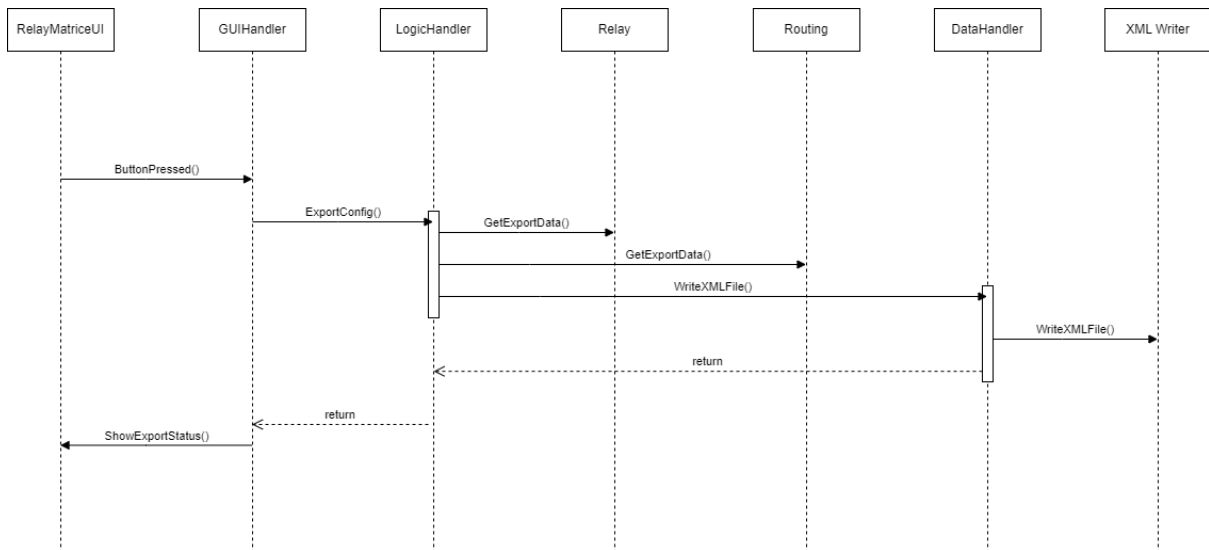


Figure 4-6 Sequence diagram iteration 2.

This sequence diagram introduces DataHandler class, which implements both the singleton pattern and the façade pattern. This is the single point of contact for the data layer.

4.3.3 Iteration 3

Section 8 and 9 of the FDUCD can be seen below. The full document can be seen in Appendix C.

Use case name: Import XML configuration file into the software.

Scope: Configuration Tool

Level: User goal

Primary actor: User

Stakeholders: Owner

Preconditions: Software started. Relay matrix window opened. XML configuration file exists.

Success guarantee: Import data from a valid XML configuration file.

Main success scenario:

1. Add valid path into the import path field.
2. Press the import button.
3. Read XML file.
4. Convert XML data to software format.
5. Show imported data to the user.

Extensions:

2a Invalid path.

- 2b Tell user of invalid path.
- 2c Go to step 1.
- 3a Invalid file type
- 3b Tell user of error.
- 3c Go to step 1.
- 5a Wrong data format/missing data
- 5b Tell user of invalid data.
- 5c Go to step 1.

Import action is started by the user. Once the import path is entered, and the import button is pressed. The software finds the XML file and reads all the data contained in the XML file. This data is then converted into the format of the software and presented back to the user.

SSD for iteration 3 can be seen in appendix D.

Sequence diagram for iteration 3 can be seen in Figure 4-7. This process uses several of the classes identified and developed in iteration 2. Some additional classes are used to read the XML file, and to parse the XML data into the format needed by the software. Once this process is complete, the data is sent back up to the GUI, for the user to see the updated status of the software. The XML files are on the same format as the files generated by the software. An example of these files can be found in the result part of the report, chapter 5.3.

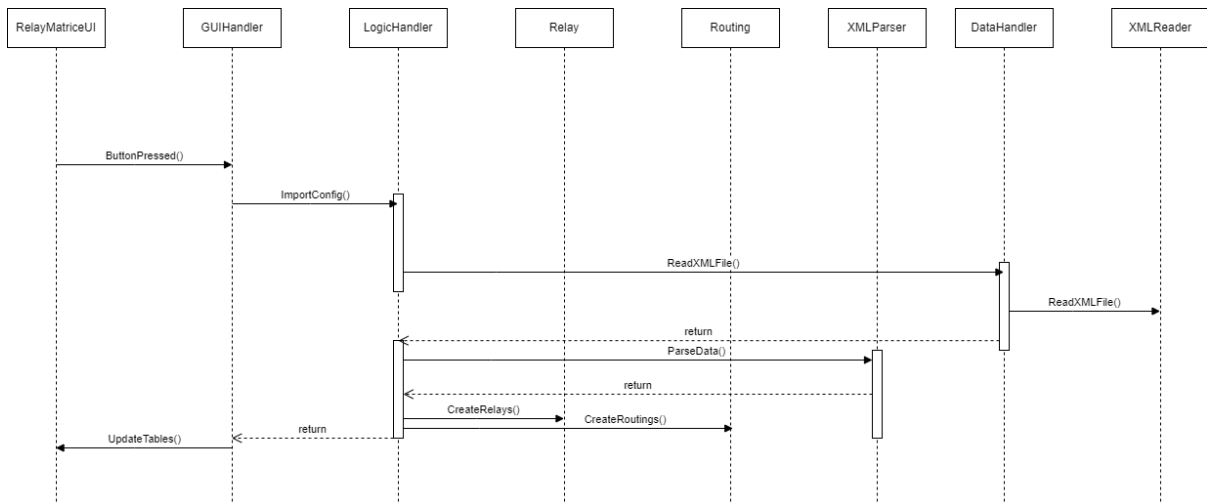


Figure 4-7 Sequence diagram iteration 3.

4.3.4 Iteration 4

The FDUCD for iteration 4 can be seen below.

Main success scenario:

1. Add valid path into the import path field.

4 Analysis, design, and implementation

2. Press the import button.
3. Read DXF file.
4. Convert DXF data to software format.
5. Show imported data to the user.

Extensions:

- 2a Invalid path.
- 2b Tell user of invalid path.
- 2c Go to step 1.
- 3a Invalid file type.
- 3b Tell user of error.
- 3c Go to step 1.
- 5a Wrong data format/missing data
- 5b Tell user of invalid data.
- 5c Go to step 1.

The process is very similar to iteration 3, which handled import of XML files. The DXF file are however very different in design, so reading them requires different operations seen in the sequence diagram.

Sequence diagram for iteration 4 can be seen in Figure 4-8. The diagram uses a DXFReader to read the file and a DXFParser to convert the data into Relays. The converted data is sent back to the GUI, for the user to see updated data.

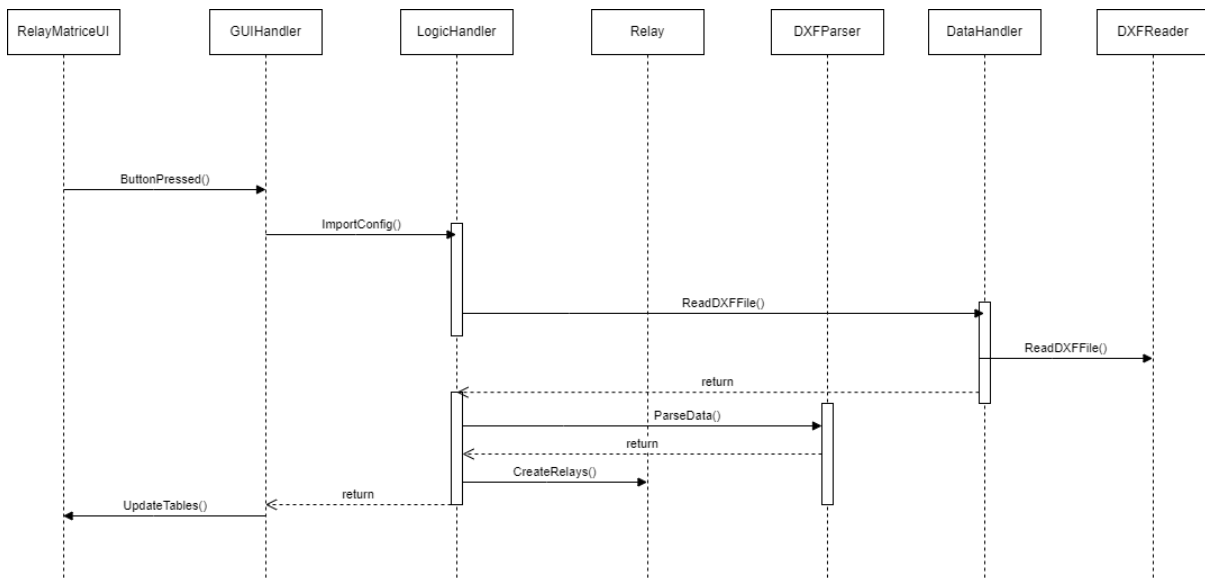


Figure 4-8 Sequence diagram iteration 4.

4.4 System architecture

The software is separated into projects. The software is configured for the addition of projects for additional modules. Currently the solution has a project for the MainWindow, another one for RelayMatrices and a last one for SharedLibraries. The projects can be seen in Figure 4-9.

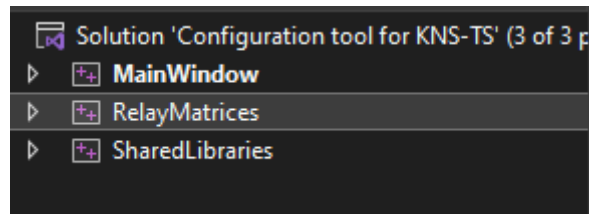


Figure 4-9 Project architecture.

The system architecture is constructed as a 3-layer architecture. It consists of a UI layer, a business logic layer, and a data access layer. This design helps create a good code structure and can allow for changes later in the software lifecycle. By separating the layers, any layer can be changed independently of the others. This would allow the software to change user interface tools without touching any of the other layers.

The UI layer and the logic layer is contained in the separate projects. Data access is handled in SharedLibraries.

The most important classes can be seen in the class diagram, Figure 4-10. Zoomed in figures for easier reading can be found below, Figure 4-11, Figure 4-12, and Figure 4-13.

4 Analysis, design, and implementation

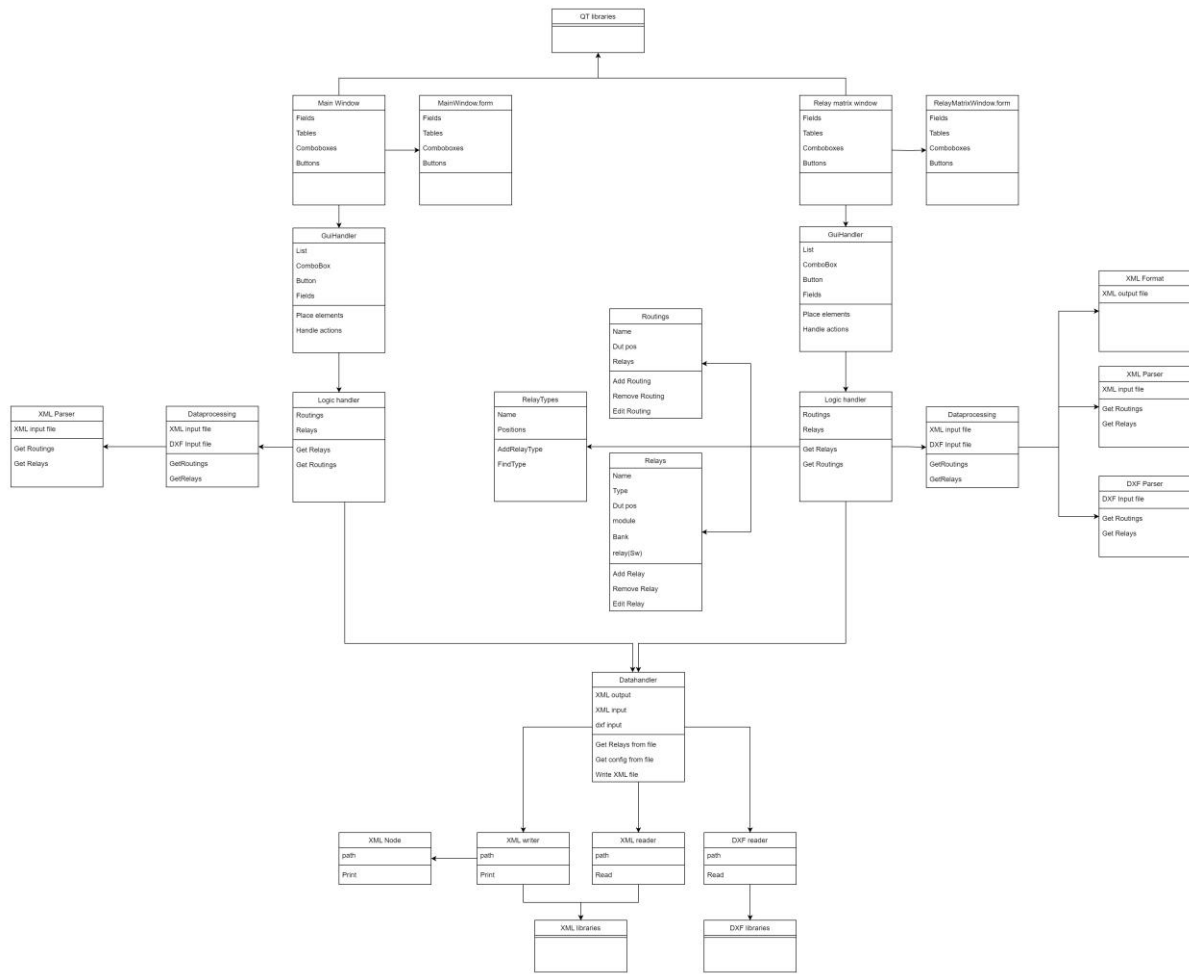


Figure 4-10 Class diagram.

4 Analysis, design, and implementation

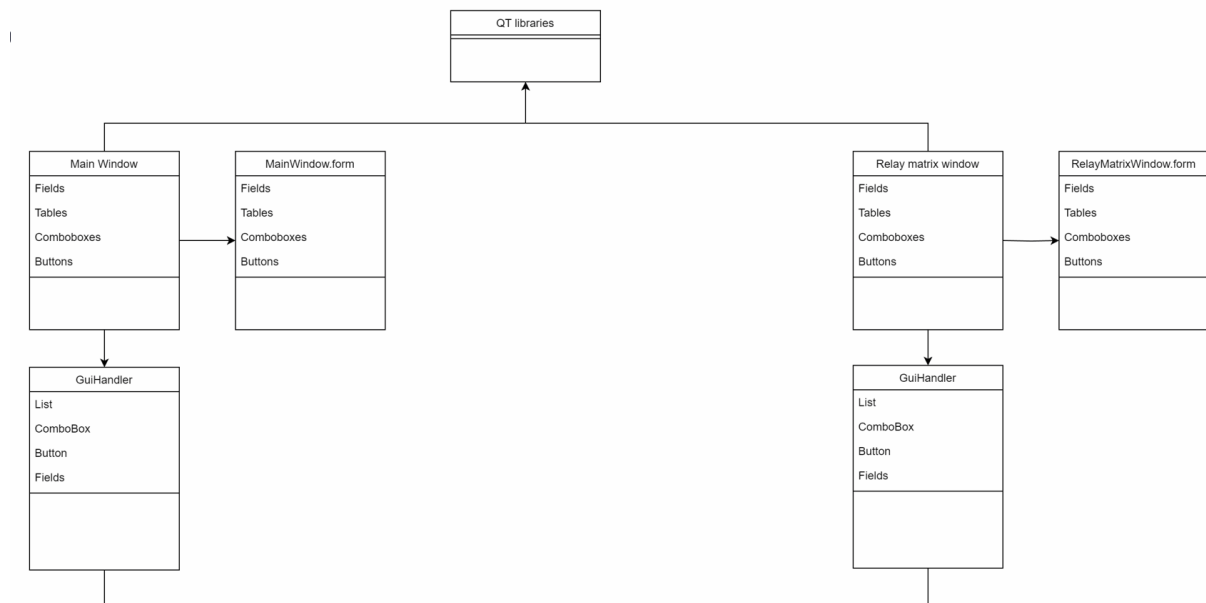


Figure 4-11 UI layer class diagram.

In the UI layer each project has a .form file. This file has all information about the visual presentation of the software. All elements are placed in these files. More information about these files can be found in chapter 4.5.

Every form file is implemented with a class. This class implements all the user interface logic and QT event handlers. These classes are also responsible for using the QT libraries.

Every project has its own GuiHandler class. This class communicates with the logic layer, via the LogicHandler class.

4 Analysis, design, and implementation

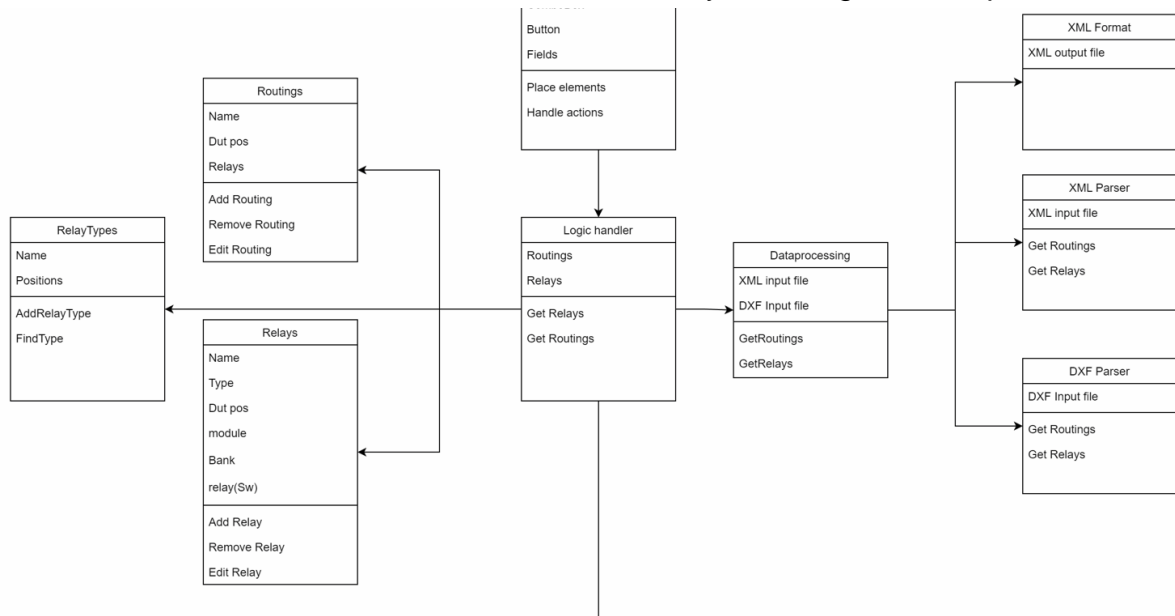


Figure 4-12 Relay matrix logic layer class diagram.

The business logic layer entry point is the LogicHandler class. Every project will have one of these classes. The logic layer contains the main data and logic of the system. Relays, routings and relay type data is stored in separate classes. All these classes have logic for validating data. Any error in the validation will invalidate the object, and an error message will be sent to the user interface layer, to be shown to the user.

Data processing is also done in this layer. Both parsing of input data and formatting of XML data. When writing to XML data, the format is done in the XMLFormat class and sent via the logicHandler to the data layer. In the data layer the actual writing is done, and the result is sent back up to the ui layer.

4 Analysis, design, and implementation

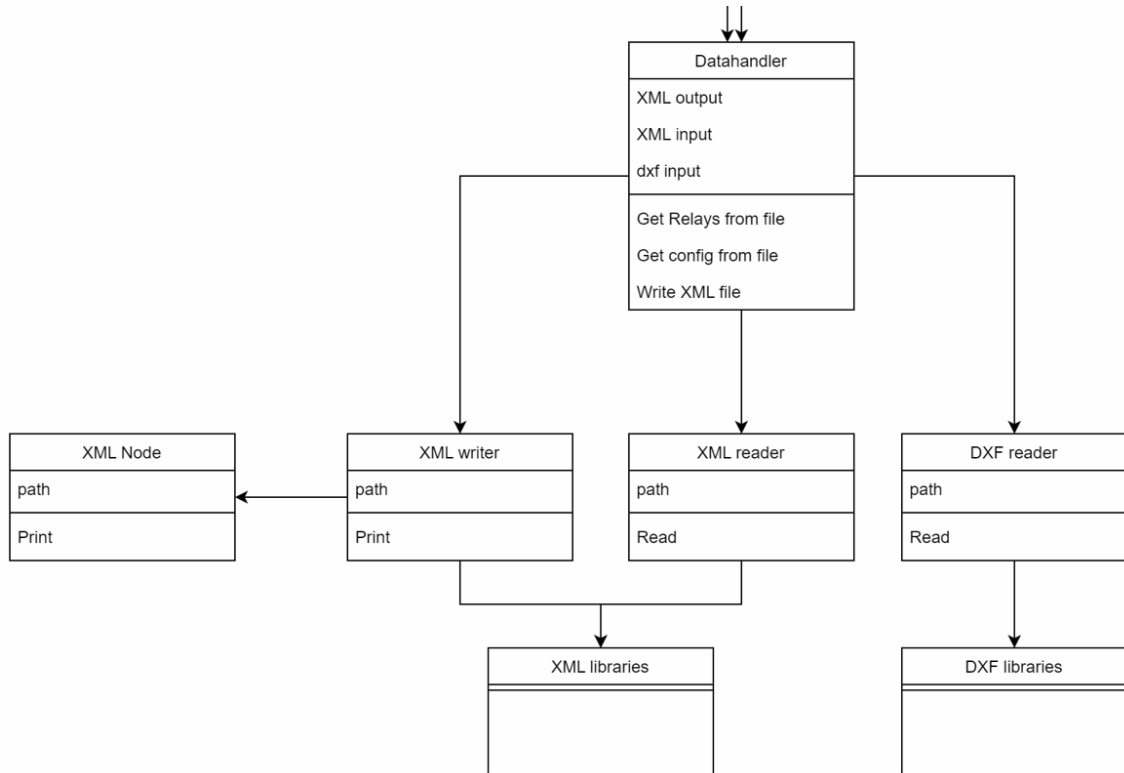


Figure 4-13 Data layer class diagram.

The Data layer has an entry point named DataHandler. This class is the only communication with the other layers. The data communication in the software consists of XML writing and reading, and DXF reading. All these have different classes to handle the functionality.

The software uses 3rd party libraries for both XML and DXF file handling. These are implemented in the XML and DXF classes respectively.

All of these classes are located in the shared libraries. This means that all of these classes are meant to be used by every module added to the software in later versions. This is done by making sure that every class returns all relevant data, and by parsing that data in the relevant project. In the existing code the RelayMatrixParser is responsible for parsing the XML data and extracting relevant data.

4.5 Implementation

Figure 4-14 shows a flow chart of typical usage of the system. It goes from starting the software, choosing the relay matrix window, adding a configuration, printing the configuration to XML and closing the window to return to the main menu.

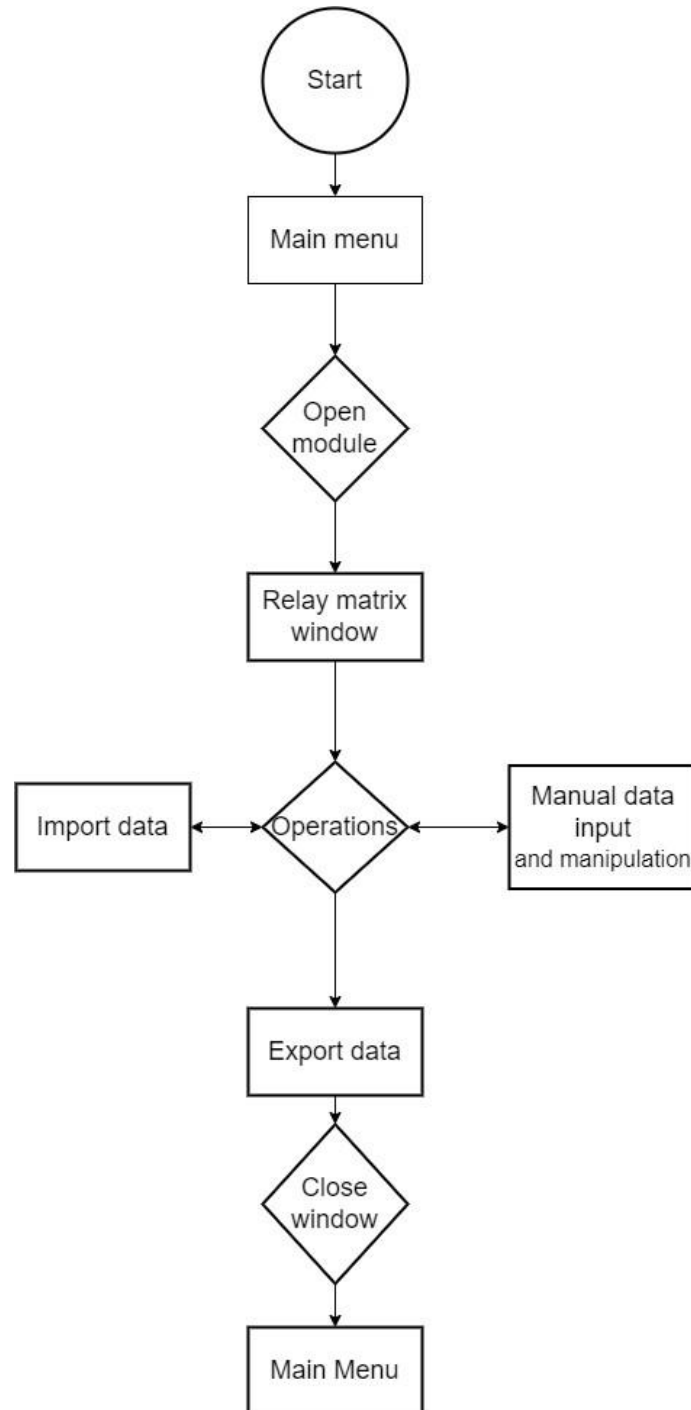


Figure 4-14 Flow chart of typical use.

4 Analysis, design, and implementation

The QT implementation is done by using a visual studio extension called QT VS tools [8]. This tool gives the user a graphical design tool for QT in Visual Studio, allow drag and drop of elements. A screenshot of the plugin can be seen in Figure 4-15.

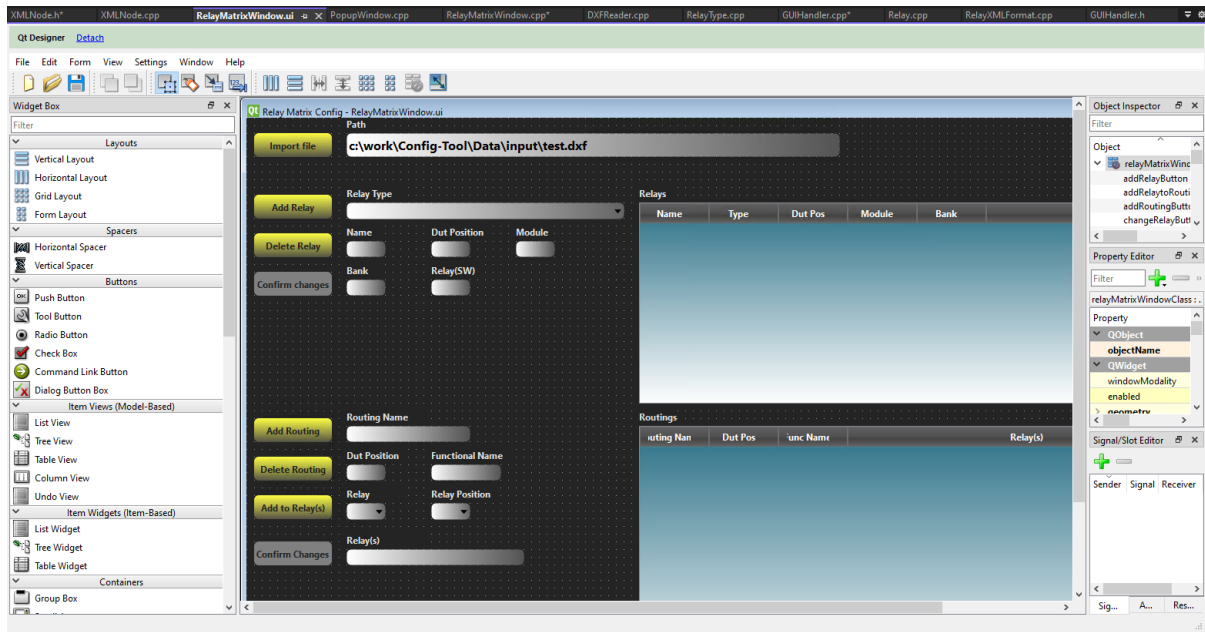


Figure 4-15 QT VS tool.

This graphical design is stored in .form files. These files use the same syntax as XML, to create objects. These objects have extensive information, like position and stylesheet stored in these files. When building the source code, these files are read and converted into header files, and compiled using a c++ compiler. An example of the syntax in one of these .form files can be seen in Appendix F. These files grow very large, so a full file example does not fit in this report. This tool allowed for effective and precise design of the user interface.

QT eventhandlers are called signals and slots. These are created in pairs to allow for the events specified by the developer to be handled. A set of default signals are made by QT to handle the standard actions. The event handler for RelayMatrixWindow class can be seen in Figure 4-16. An example of a slot, the function called when the signal is sent, can be seen in Figure 4-17. When a row in the table is selected, an itemSelectionChanged signal is sent by QT. This is connected to the slot called SelectedRelaySlot. This means that this function will be called whenever the signal is emitted. In this example, the function gets the data from the row, and fills it into the appropriate text field next to the table.


```

//Setup event handlers
void
RelayMatrixWindow::ConnectSlots()
{
    connect(ui.relayTable, SIGNAL(itemSelectionChanged()), this, SLOT(SelectedRelaySlot()));
    connect(ui.routingTable, SIGNAL(itemSelectionChanged()), this, SLOT(SelectedRoutingSlot()));

    connect(ui.importButton, SIGNAL(clicked()), this, SLOT(ImportConfigSlot()));
    connect(ui.exportButton, SIGNAL(clicked()), this, SLOT(ExportConfigSlot()));

    connect(ui.addRelayButton, SIGNAL(clicked()), this, SLOT(AddRelaySlot()));
    connect(ui.deleteRelayButton, SIGNAL(clicked()), this, SLOT(DeleteRelaySlot()));
    connect(ui.changeRelayButton, SIGNAL(clicked()), this, SLOT(UpdateRelaySlot()));

    connect(ui.addRoutingButton, SIGNAL(clicked()), this, SLOT(AddRoutingSlot()));
    connect(ui.deleteRoutingButton, SIGNAL(clicked()), this, SLOT(DeleteRoutingSlot()));
    connect(ui.addRelaytoRoutingButton, SIGNAL(clicked()), this, SLOT(AddRelayRoutingSlot()));
    connect(ui.changeRoutingButton, SIGNAL(clicked()), this, SLOT(UpdateRoutingSlot()));

    connect(ui.relayNameEdit, SIGNAL(textEdited(const QString&)), this, SLOT(RelayDataChangedSlot(const QString&)));
    connect(ui.relayDutPosEdit, SIGNAL(textEdited(const QString&)), this, SLOT(RelayDataChangedSlot(const QString&)));
    connect(ui.relayBankEdit, SIGNAL(textEdited(const QString&)), this, SLOT(RelayDataChangedSlot(const QString&)));
    connect(ui.relayModuleEdit, SIGNAL(textEdited(const QString&)), this, SLOT(RelayDataChangedSlot(const QString&)));
    connect(ui.relaySWEdit, SIGNAL(textEdited(const QString&)), this, SLOT(RelayDataChangedSlot(const QString&)));
    connect(ui.relayTypeComboBox, SIGNAL(currentIndexChanged(int)), this, SLOT(TypeComboboxChangedSlot(int)));
    connect(ui.relayComboBox, SIGNAL(currentIndexChanged(int)), this, SLOT(relayComboboxChangedSlot(int)));

    connect(ui.routingNameEdit, SIGNAL(textEdited(const QString&)), this, SLOT(RoutingDataChangedSlot(const QString&)));
    connect(ui.routingDutPosEdit, SIGNAL(textEdited(const QString&)), this, SLOT(RoutingDataChangedSlot(const QString&)));
    connect(ui.nameEdit, SIGNAL(textEdited(const QString&)), this, SLOT(RoutingDataChangedSlot(const QString&)));
    connect(ui.relayRoutingEdit, SIGNAL(textEdited(const QString&)), this, SLOT(RoutingDataChangedSlot(const QString&)));
}

```

Figure 4-16 Event handlers in RelayMatrixWindow

```

162 //Handle select row in relay table
163 void
164 RelayMatrixWindow::SelectedRelaySlot()
165 {
166     QList<QTableWidgetItem*> items = ui.relayTable->selectedItems();
167     if (items.count() == 0)
168     {
169         return;
170     }
171     QString name = items.at(0)->text();
172     ui.relayNameEdit->setText(name);
173
174     //Find combobox index from string
175     for (auto i = 0; i < ui.relayTypeComboBox->count(); i++)
176     {
177         QString comboBoxText = ui.relayTypeComboBox->itemText(i);
178         QString type = items.at(1)->text();
179         if (comboBoxText == type)
180         {
181             ui.relayTypeComboBox->setCurrentIndex(i);
182             break;
183         }
184     }
185
186     QString dutPos = items.at(2)->text();
187     ui.relayDutPosEdit->setText(dutPos);
188
189     QString module = items.at(3)->text();
190     ui.relayModuleEdit->setText(module);
191
192     QString bank = items.at(4)->text();
193     ui.relayBankEdit->setText(bank);
194
195     QString relaySW = items.at(5)->text();
196     ui.relaySWEdit->setText(relaySW);
197
198     ui.changeRelayButton->setDisabled(true);
199 }
200

```

Figure 4-17 Slot for selecting a table entry.

4 Analysis, design, and implementation

3rd part libraries are used for both XML and DXF files. The XML library is called Libxml2 [9]. The library is written in C, and it has multilanguage support. It's a free software available under the MIT license.

To interface this library, a set of classes was made in the configuration tool software. These are a XMLReader, a XMLWriter, and a XMLNode, as seen in Figure 4-18.

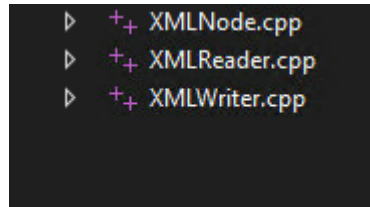


Figure 4-18 XML interface.

The reader and writer calls functions needed for their respective functions, while the node is used to package data into objects after it's been read.

A code example for the writer can be seen Figure 4-19. The printXML function goes through the XMLTagData vector created by the formater. Depending on the tag type, different functions are called. Each one of these functions writes an element to the XML file. For example, the WriteSingleTag function shown in the example. It will create a start element of type tag, and the end element for the same tag. If the value of tag is "test" the XML file will look like this; <test> </test>.

```
83 int
84 XMLWriter::printXML(std::vector<XMLTagData> output)
85 {
86     for (XMLTagData tag : output)
87     {
88         switch (tag.tagType)
89         {
90             case StartTag:
91                 WriteStartTag(tag.name);
92                 break;
93             case EndTag:
94                 WriteEndTag();
95                 break;
96             case SingleTag:
97                 WriteSingleTag(tag.name);
98                 break;
99             case ValueTag:
100                 WriteValueTag(tag.name, tag.value);
101                 break;
102             case comment:
103                 WriteComment(tag.value);
104                 break;
105             case noTag:
106                 default:
107                     return 0;
108                     break;
109         }
110     }
111     int signs = xmlTextWriterEndDocument(fWriter);
112     return signs;
113 }
114
115 void XMLWriter::WriteSingleTag(std::string tag)
116 {
117     xmlTextWriterStartElement(fWriter, BAD_CAST tag.c_str());
118     xmlTextWriterEndElement(fWriter);
119 }
120
```

Figure 4-19 Algorithm for writing XML output.

4 Analysis, design, and implementation

The DXF library used is called DxfLib [10]. It's an open-source software available under the General Public license. The library classifies the data from the DXF file, allowing the configuration tool software to filter out most of the data in the large DXF file. This is done in a class called DXFReader. The filtered data is sent to the DXFParser located in the logic layer. This parser checks the data and tries to use it to create valid relays.

Some Grasp [1, p105], General Responsibility Assignment Software Patterns, are used to create a better object-oriented design.

The controller pattern describes which classes are responsible for each use case. In this software, the UI layer starts every use case. This makes the relay matrix window the start and end point of every use case. It does however not do much more than start and end the use case. The main functionality is placed in the logic or data layer.

The information expert pattern tells us to add responsibilities where the information needed already exists. This is for example used by error checking of relays is done in the relay class, seen in Figure 4-20.

```
146
147 void
148 Relay::SetBank(std::string newBank)
149 {
150     try
151     {
152         SetBank(std::stoi(newBank));
153     }
154     catch (std::invalid_argument const& ex)
155     {
156         isValid = false;
157         errorString.append("Invalid Bank \n");
158     }
159 }
160
161 void
162 Relay::SetRelaySW(std::string newRelaySW)
163 {
164     int newVal;
165     try
166     {
167         newVal = std::stoi(newRelaySW);
168     }
169     catch (std::invalid_argument const& ex)
170     {
171         isValid = false;
172         errorString.append("Invalid Relay(SW) \n");
173         return;
174     }
175     SetRelaySW(newVal);
176 }
177
```

Figure 4-20 Set functions with validation checks

High cohesion pattern tells us how to keep objects from growing too big and confusing. By creating more classes with smaller responsibilities, high cohesion is secured.

Low coupling pattern describes dependencies in between objects. Low coupling means that a change in one object does not require a lot of changes in other objects. In this software, this is secured by having one way communication in every object interaction.

5 Results

This chapter describes the results of the work. The focus is put on showing the system as it is at the end of the development process. The first subchapter describes the main window. The next 4 subchapters will describe the implementation of the use cases. The last subchapter describe testing. The style of the user interface is made using the stylesheet of already existing software at DSS Horten, with a few alterations.

5.1 Main window

The main window has a simple user interface, shown in Figure 5-1. This is the first window the user sees when starting the software. The design of the window is meant to allow for additional configuration systems to be added, in the form of buttons. Currently only the relay matrix configuration is implemented. Clicking the Relay matrices button opens the window described in 5.2.

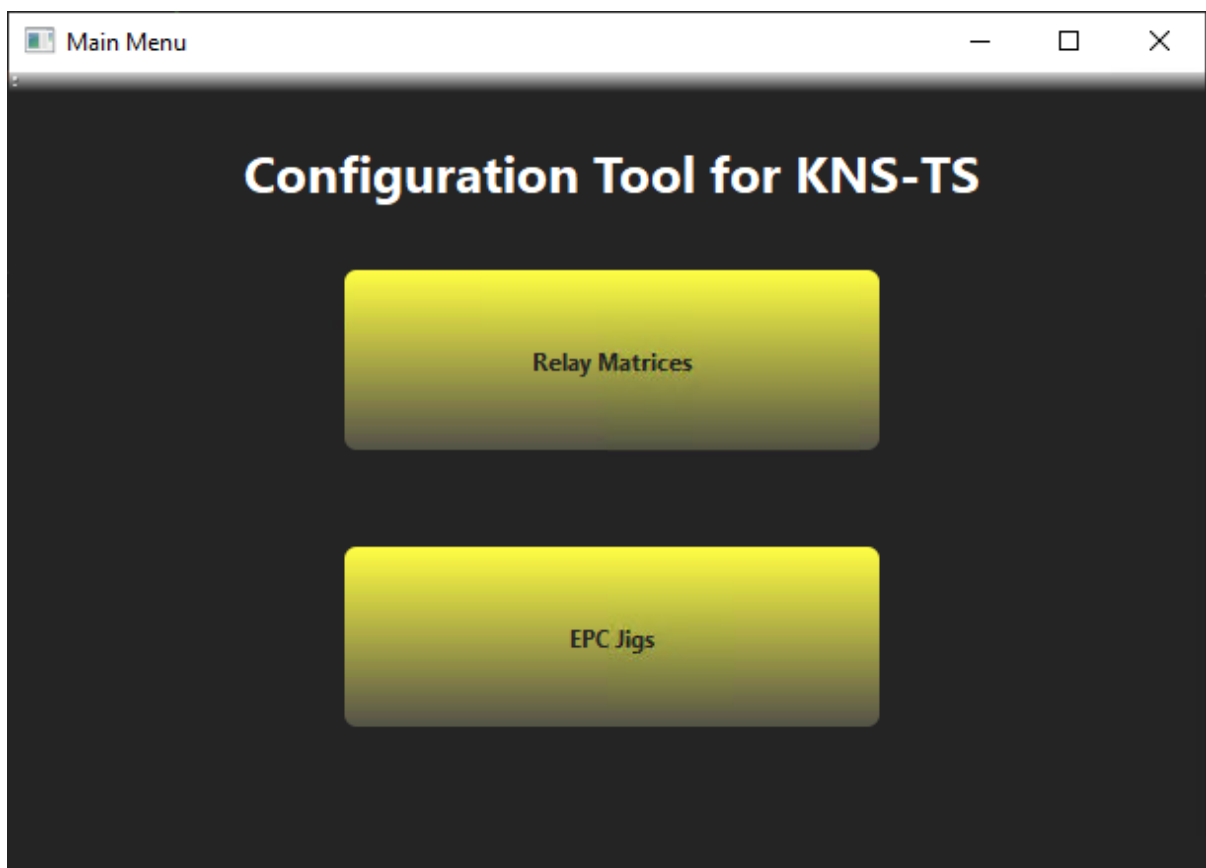


Figure 5-1 Main window.

5.2 Manual input

The first use case describes manual input of data into the system, and the showing of this data in a GUI. A screenshot of the implementation can be seen in Figure 5-2. The user interface consists of 2 tables showing the saved data of relays and routings, as well as several fields and buttons to manipulate the data in these tables. When the user exports the configuration, the data currently shown in the tables is converted to an XML file.

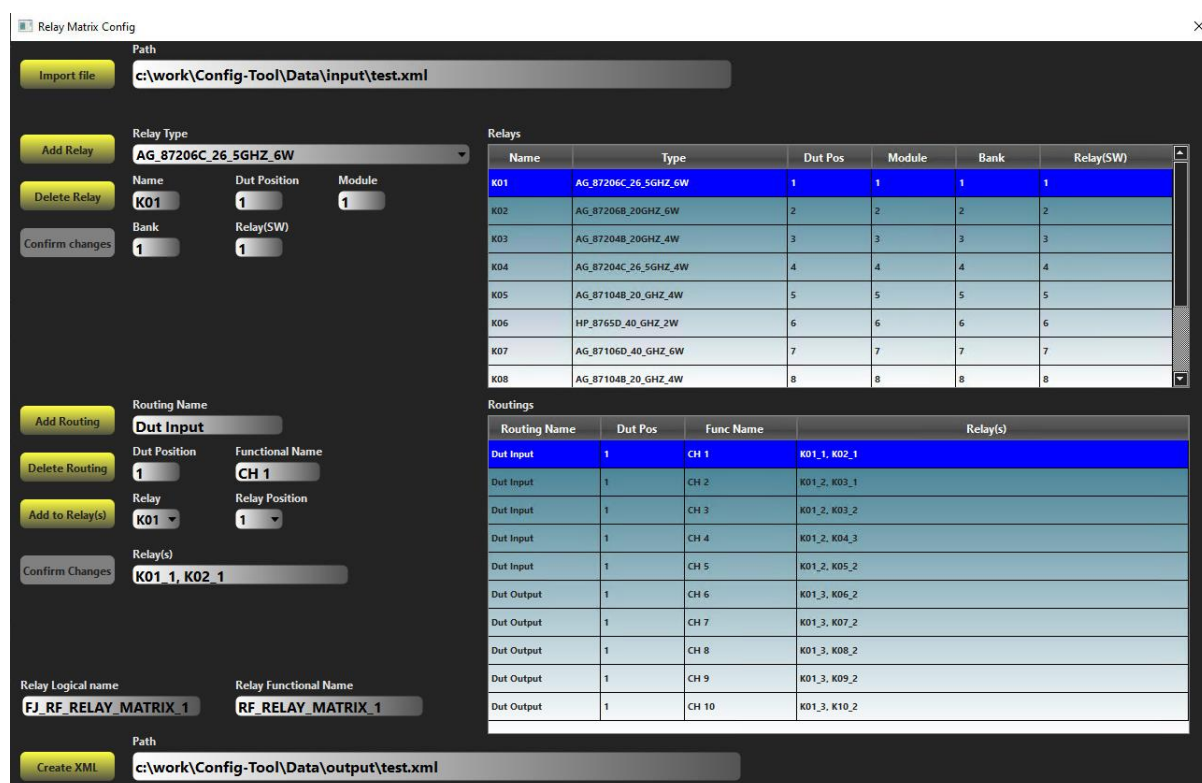


Figure 5-2 The relay matrix config user interface.

The fields used to describe the relays and routings are taken from the requirements of the system. Both sets of data have 3 of the same buttons connected to it; add, delete, and confirm changes.

The add button is used to create new elements from the values available in the text fields. That means that when the user clicks “Add Relay” button, the system takes the values from relay type, name, dut position, module, bank, and relay(SW) and tries to create a relay. The relay type combobox is populated when opening the relay matrix window by reading an XML file. How the system reads XML files will be described in chapter 5.4.

In Figure 5-3 the code for adding a relay can be seen. In each set function the values are validated. If any value is invalid the relay is rejected and an error message is shown, as shown in Figure 5-4.

```

E Relay::Relay(std::vector<std::string> values)
{
E   if (values.size() != 6)
   {
       isValid = false;
       errorString.append("Size of value vector incorrect \n");
       return;
   }
   SetName(values[0]);
   SetType(values[1]);
   SetDutPos(values[2]);
   SetModule(values[3]);
   SetBank(values[4]);
   SetRelaySW(values[5]);
}

```

Figure 5-3 Code for creating a new relay.

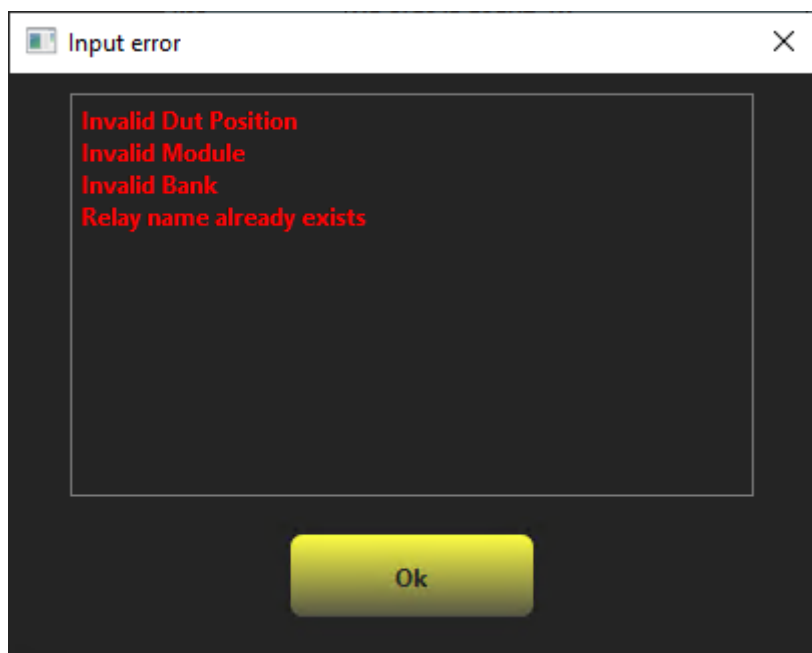


Figure 5-4 Error message example.

The delete button removes a row from the table in the user interface. The button also calls a function to delete the corresponding entry from the vector in the logic layer. When a row is deleted, the row below becomes selected. If the delete button is pressed without any row being selected an error message, as seen in Figure 5-5, is shown.

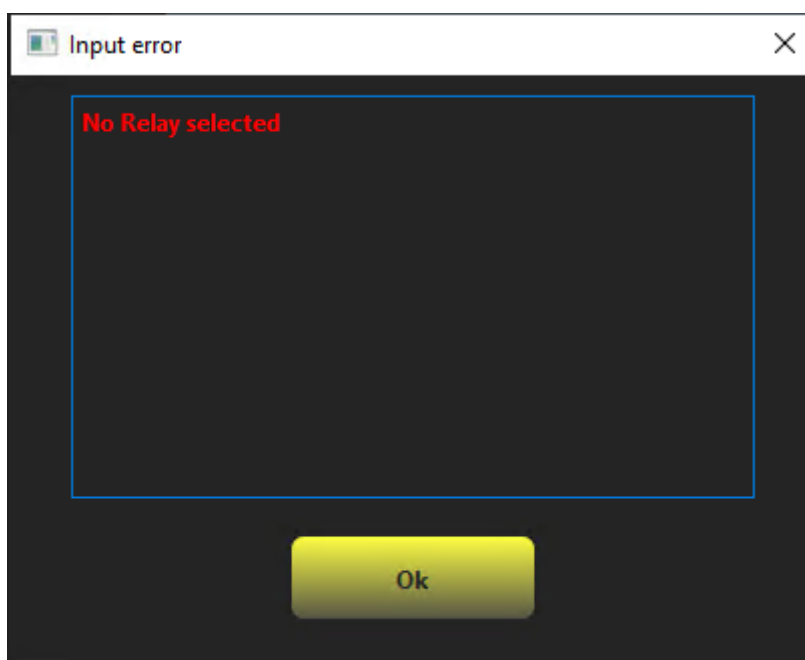


Figure 5-5 No row selected when deleting.

The confirm changes button is used to update a row with new data. When the user selects a row, the data from this row is shown in the text fields next to the table. The button is inactive until a row is selected and any of the values are altered. Once a text field value is altered, or a combobox index is changed, the button becomes active. The new data is validated by using the same functions as the add functionality.

In the relay section there is an extra button called “Add to Relay(s)”, seen in Figure 5-6. This button is used to add values to the field Relay(s). The available options in the relay combobox are the relays currently stored in the system. Based on the type of the relay, the valid positions are added to the relay position combobox. When the button is clicked, a string on the format, name_position, is added to relay(s). It’s also possible to directly edit the Relay(s) field if the user is experienced and prefer this option.

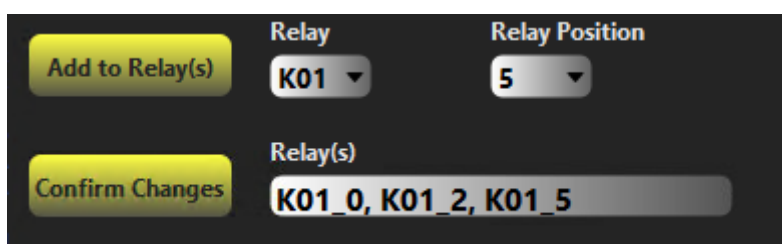


Figure 5-6 Add to relay button.

5.3 Export XML

To export the configuration made using the software to an XML file, the user has to specify a path with filename and press the create XML button. A screenshot of this is shown in Figure 5-8. Exporting overwrites any existing file with the same path and name. This is shown to the user in a message box, as seen in Figure 5-7.

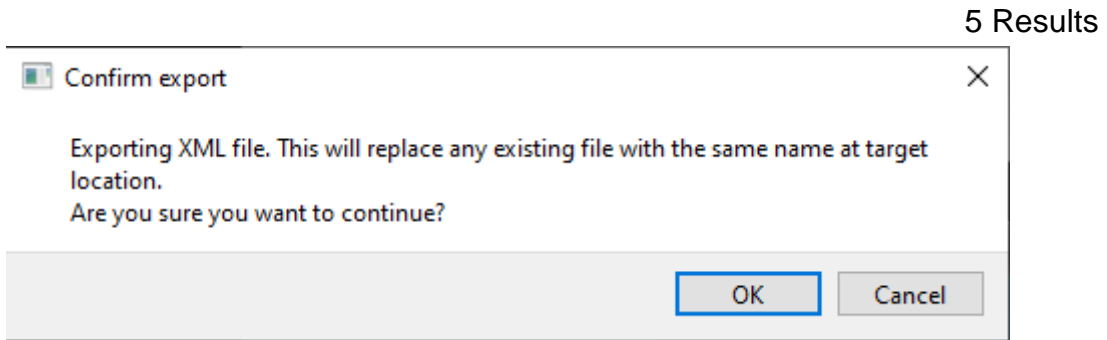


Figure 5-7 message box export warning.



Figure 5-8 Export user interface.

When exporting, the software checks if the operation was successful or not. Depending on the result, the software will either show a success label, as in Figure 5-9, or show an error message, as in Figure 5-10.



Figure 5-9 Successful export

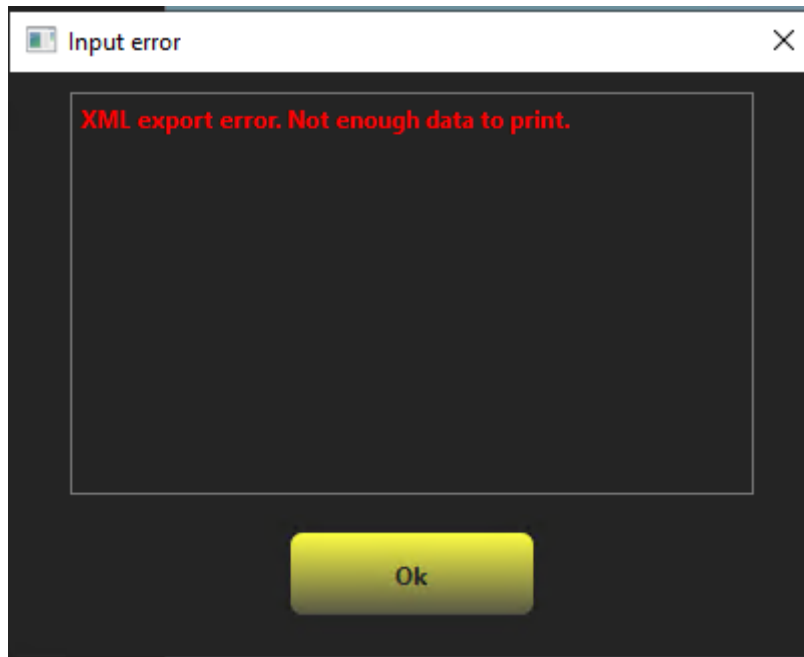


Figure 5-10 Failed export.

The export is formatted in a separate class, called RelayXMLFormat. In this class the XML format is created based on the current standard for relay matrix configuration. This also means that if the standard changes in the future, the format can be altered by only making changes in this class. A part of the format can be seen in Figure 5-11. This is the code for adding every relay to the XML sheet.

```
26
27     for (size_t i = 0; i < relays.size(); i++)
28     {
29         output.emplace_back(XMLTagData(StartTag, "relay"));
30         output.emplace_back(XMLTagData(ValueTag, "name", relays.at(i).GetName()));
31         output.emplace_back(XMLTagData(ValueTag, "type", relays.at(i).GetType()));
32
33         output.emplace_back(XMLTagData(StartTag, "controller"));
34         output.emplace_back(XMLTagData(ValueTag, "functional_name", "F_Relay_Controller"));
35         output.emplace_back(XMLTagData(ValueTag, "dut_pos", std::to_string(relays.at(i).GetDutPos())));
36         output.emplace_back(XMLTagData(ValueTag, "module", std::to_string(relays.at(i).GetModule())));
37         output.emplace_back(XMLTagData(ValueTag, "bank", std::to_string(relays.at(i).GetBank())));
38         output.emplace_back(XMLTagData(ValueTag, "relaysw", std::to_string(relays.at(i).GetRelaySW())));
39         output.emplace_back(XMLTagData(EndTag, ""));
40     }
```

Figure 5-11 Code for relay XML format.

This code creates a vector of XML nodes that are sent to the XML writer, and then written to a XML file.

An example of a created XML files can be found in Appendix J. The start of the file, with data for 1 relay can be seen in Figure 5-12. Everything between the relay tags is data configurable by the software.

```

1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <test_setup>
3    <jigs>
4      <!-->
5      <Relay_matrix>
6        <matrix>
7          <logical_name>RF_RELAY_MATRIX_1</logical_name>
8          <port>
9            <functional_name>FJ_RF_RELAY_MATRIX_1</functional_name>
10           <dut_pos>1</dut_pos>
11          </port>
12        </matrix>
13      <relay>
14        <name>K01</name>
15        <type>AG_87206C_26_5GHZ_6W</type>
16        <controller>
17          <functional_name>F_Relay_Controller</functional_name>
18          <dut_pos>1</dut_pos>
19          <module>1</module>
20          <bank>1</bank>
21          <relaysw>1</relaysw>
22        </controller>
23        <relay_pos>
24          <name>K01_0</name>
25          <relay_position>0</relay_position>
26        </relay_pos>
27        <relay_pos>
28          <name>K01_1</name>
29          <relay_position>1</relay_position>
30        </relay_pos>
31        <relay_pos>
32          <name>K01_2</name>
33          <relay_position>2</relay_position>
34        </relay_pos>
35        <relay_pos>
36          <name>K01_3</name>
37          <relay_position>3</relay_position>
38        </relay_pos>
39        <relay_pos>
40          <name>K01_4</name>
41          <relay_position>4</relay_position>
42        </relay_pos>
43        <relay_pos>
44          <name>K01_5</name>
45          <relay_position>5</relay_position>
46        </relay_pos>
47        <relay_pos>
48          <name>K01_6</name>
49          <relay_position>6</relay_position>
50        </relay_pos>
51      </relay>

```

Figure 5-12 Start of XML configuration file.

5.4 Import XML files

Importing existing configurations are handled at the very top of the user interface, see Figure 5-13. The path in the picture must consist of the entire path with file extension. The software automatically detects if the file is a XML file or a DXF file. Any other file format will produce an error message, as seen in Figure 5-14.



Figure 5-13 Import button and path

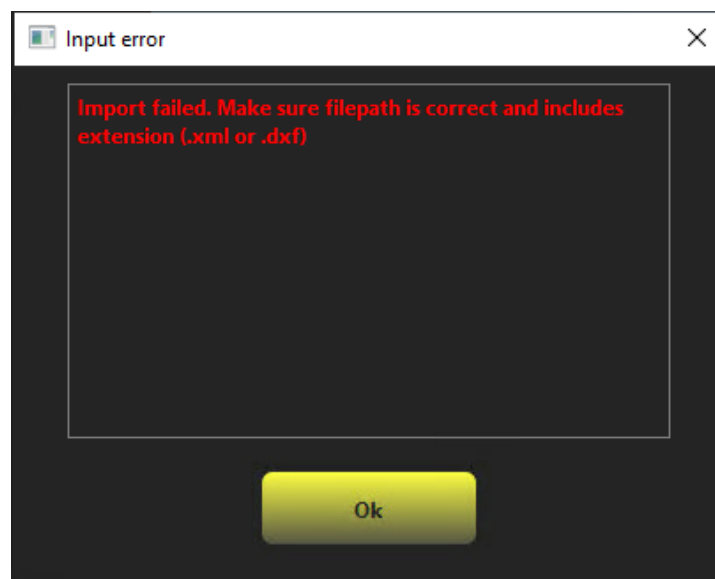


Figure 5-14 Import error

The XML file is read by the XMLReader class. This class uses a 3rd party library to read nodes, the nodes identified are sent to the parser in the logic layer. This parser is called RelayXMLParser. This parser looks for the data needed to create valid relays and routings in the software. After going through the entire XML file, the created relays are stored, any errors found are shown to the user, see Figure 5-16, and the tables in the user interface are updated with the new data. Any previously stored data is replaced when an import is done. This is presented to the user in a message box, as seen in Figure 5-17. An example of a successful import from the test.xml file can be seen in Figure 5-15.

Relays					
Name	Type	Dut Pos	Module	Bank	Relay(SW)
K01	AG_87206C_26_5GHZ_6W	1	1	1	1
K02	AG_87206B_20GHZ_6W	2	2	2	2
K03	AG_87204B_20GHZ_4W	3	3	3	3
K04	AG_87204C_26_5GHZ_4W	4	4	4	4
K05	AG_87104B_20_GHZ_4W	5	5	5	5
K06	HP_8765D_40_GHZ_2W	6	6	6	6
K07	AG_87106D_40_GHZ_6W	7	7	7	7
K08	AG_87104B_20_GHZ_4W	8	8	8	8

Routings			
Routing Name	Dut Pos	Func Name	Relay(s)
Dut Input	1	CH 1	K01_1, K02_1
Dut Input	1	CH 2	K01_2, K03_1
Dut Input	1	CH 3	K01_2, K03_2
Dut Input	1	CH 4	K01_2, K04_3
Dut Input	1	CH 5	K01_2, K05_2
Dut Output	1	CH 6	K01_3, K06_2
Dut Output	1	CH 7	K01_3, K07_2
Dut Output	1	CH 8	K01_3, K08_2
Dut Output	1	CH 9	K01_3, K09_2
Dut Output	1	CH 10	K01_3, K10_2

Figure 5-15 Import from test.xml.

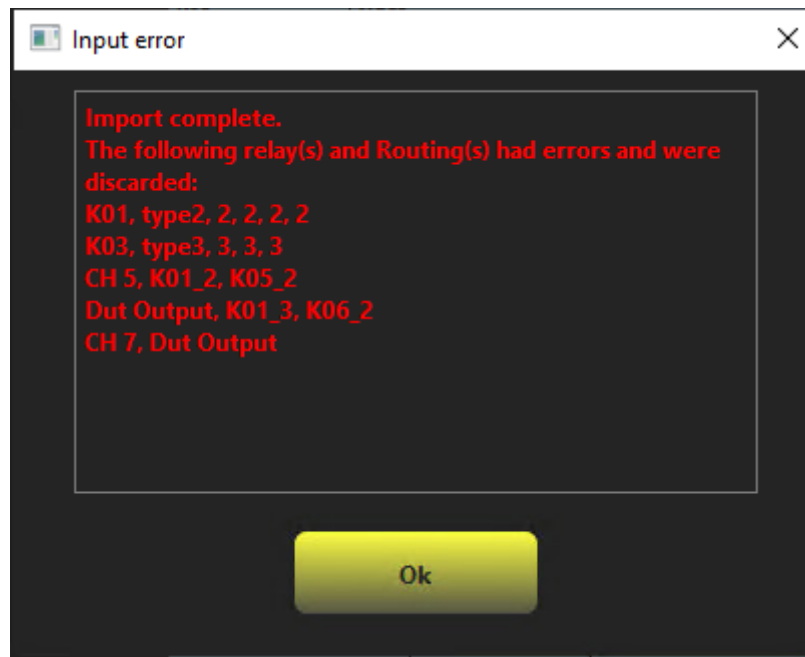


Figure 5-16 Example of errors during import.

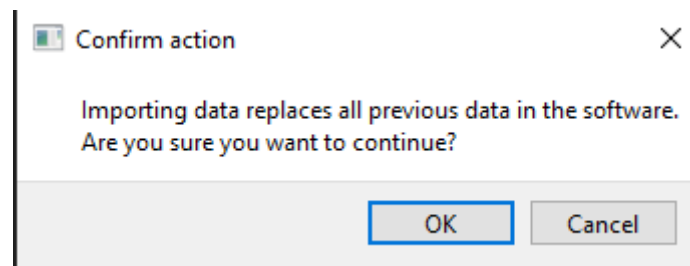


Figure 5-17 Confirmation window import.

5.5 Import DXF files

Importing DXF files are very similar to importing XML file. The user simply adds the path and extension to a DXF file before pressing import. In the software, the extension is recognized, and the software creates a DXFReader object. This reader uses a 3rd party library called DxfLib to search through the file and find all the relevant strings from the DXF file and sends them to the parser. The parser then goes through all the filtered data and tries to create valid relays. This can be done because DSS has a naming convention on both relay name and relay type in the DXF files. The naming convention has relay names containing needed information on modules, banks, and switches. An example would be “K01_M1_B1_S1”. This means that relay “K01” is connected in module 1, bank 1 and switch 1. The relay types are written using a unique substring of the full relay name. To connect the type to a specific relay, the name is also included. An example would be “K01_87104b”. From these 2 string a valid relay can be created.

5 Results

No routing data can be found from the DXF files. The valid relays created are stored in the system and the user interface tables are updated. Any data discarded in the parser is shown in an error message to the user. This data could be names not following the naming conventions, or other string be very similar to the data the system looks for. Either way this is considered useful information for the user of the system.

Figure 5-18 shows an example of the DXF file. These files become very big. The lines containing the relevant data for this software can be seen on line 140920 and 140936.

```
140915 30
140916 0.0
140917 40
140918 1.351063829787234
140919 1
140920 K01_87104B
140921 0
140922 TEXT
140923 5
140924 75E
140925 8
140926 SHEET
140927 10
140928 165.8070177430802
140929 20
140930 243.731914893617
140931 30
140932 0.0
140933 40
140934 1.351063829787234
140935 1
140936 K01_M1_B3_S1
140937 0
140938 POLYLINE
140939 5
140940 75F
140941 8
140942 SHEET
140943 66
```

Figure 5-18 DXF file example.

Figure 5-19 shows a successful import from the test DXF file.

Relays					
Name	Type	Dut Pos	Module	Bank	Relay(SW)
K01	AG_87104B_20_GHZ_4W	1	1	3	1
K02	AG_87106B_20_GHZ_6W	1	1	2	1
K03	AG_87104B_20_GHZ_4W	1	2	4	1
K04	HP_8765B_20_GHZ_2W	1	1	4	1
K11	AG_87106B_20_GHZ_6W	1	2	2	1
K12	AG_87106B_20_GHZ_6W	1	2	3	1
K13	HP_8765B_20_GHZ_2W	1	1	4	2
K14	HP_8765B_20_GHZ_2W	1	1	4	3

Routings			
Routing Name	Dut Pos	Func Name	Relay(s)

Figure 5-19 Import from DXF file.

Figure 5-20 shows an import of DXF file with some errors. These errors can be duplicate names, missing identifiers, or invalid data values.

Relays

Name	Type	Dut Pos	Module	Bank	Relay(SW)
K02	AG_87106B_20_GHZ_6W	1	1	2	1
K03	AG_87104B_20_GHZ_4W	1	2	4	1
K04	HP_8765B_20_GHZ_2W	1	1	4	1
K11	AG_87106B_20_GHZ_6W	1	2	2	1
K12	AG_87106B_20_GHZ_6W	1	2	3	1
K13	HP_8765B_20_GHZ_2W	1	1	4	2
K15	AG_87106B_20_GHZ_6W	1	1	4	3

Routings

Input error

Import complete.
The following relay(s) and Routing(s) had errors and were discarded:
K17_M1_S4_S5
K16_1_B4_S4
K01_M1_B3_J1
K15_M2_B1_S1

Ok

Figure 5-20 Import DXF file with error.

5.6 Testing

The system has gone through different forms of testing during the development process.

As this has been a single developer project, most of the testing has been by the developer of the actual code. Every implementation is naturally followed by testing to confirm that the wanted functionality is present. This mostly includes manual unit testing, often done by stepping through the code in debug mode. This allowed the tester to confirm the values of different variables while the software is running, in addition to confirming the functionality of each function. It was decided not to spend resources on integrating a unit test library for this project, but this would be considered a good investment if the system is to go through further development.

After every iteration the use case went through integration testing. The entire use case was tested, and it was determined if the use case was successfully implemented. This also

5 Results

included a more determined error testing, where the tester was actively trying to produce errors or bugs. If the tests were considered successful, the development could move on to the next use case.

After the final use case was implemented, the software went through a system test. In this test, a test plan was created. This test plan contains a system description, and a description of the external files needed to test import functionality is described. Lastly a set of tasks, spanning the current functionality of the system, was created. The full test plan can be found in appendix E. An example of a performed test plan can be found in Figure 5-21.

Index	Description test case	Expected result	Result (Ok/fail)	Comment
	Tester: Krister Pedersen			
1	Start software and open relay matrix window	Relay matrix GUI shown	ok	
2	Import dxf: Import test file c:\work\Config-Tool\Data\input\test.dxf	A set of relays imported into system	ok	11 relays added
3	Add Relay: Name K05, Type AG_87206B_20GHZ_6W, DutPos 1, Module 1, Bank 1, Relay(SW) 1	Relay added to Relay table. No error	ok	
4	Add Relay: Name K06, Type AG_87204B_20GHZ_4W, DutPos 1, Module 1, Bank 1, Relay(SW) 2	Relay added to Relay table. No error	ok	
5	Add Relay: Name K07, AG_N1810UL_26_5GHZ_2W, DutPos 1, Module 1, Bank 2, Relay(SW) 1	Relay added to Relay table. No error	ok	
6	Error message: Try adding another Relay with name K01	Relay NOT added to Relay table. Error message shown	ok	Clear error message shown
7	Error message: Try adding Relay with empty field(s)	Relay NOT added to Relay table. Error message shown	ok	Clear error message shown
8	Error message: Try adding Relay with illegal values	Relay NOT added to Relay table. Error message shown	ok	Clear error message shown
9	Change Name on relay K07 to K08	Relay updated in Relay table. No error	ok	
10	Change data of K08 to TypeHP_8765B_20_GHZ_2W, DutPos 2, Module 2, Bank 2, Relay(SW) 2	Relay updated in Relay table. No error	ok	
11	Delete Relay K08	Relay removed from Relay table. No error	ok	
12	Add Routing: Name DUT Input, DutPos 1, ChannelName CH 1, Relay(s) K01_1, K02_1	Routing added to Routing table. No error	ok	
13	Add Routing: Name DUT Output, DutPos 1, ChannelName CH 2, Relay(s) K01_2, K02_2	Routing added to Routing table. No error	ok	
14	Add Routing: Name Temp, DutPos 1, ChannelName CH 2, Relay(s) K01_1	Routing added to Routing table. No error	ok	
15	Change routing Temp: add relay k02, position 2 to relay(s) string using comboboxes and "add to r	Routing updated in Routing table. No error	ok	
16	Error message: Try adding Routing with empty field(s)	Routing NOT added to Routing table. Error message shown	ok	
17	Error message: Try adding Routing with illegal values	Routing NOT added to Routing table. Error message shown	ok	
18	Delete routing Temp	Routing removed from Routing table. No error	ok	
19	Create XML: Export xml config to a path of your choosing. Call the file test.xml	xml file created. Result label shown	ok	
20	Import XML: Import test file c:\work\Config-Tool\Data\input\test.xml	Previous data replaced with new data	ok	

Figure 5-21 Test table.

The test result shows all functionalities to be working.

6 Future development

This chapter describes future development of the software. It will describe how additional modules can be implemented. This will be done by using the planned electrical power consumption module as an example.

6.1 Implementing epc configuration

The company, DSS, has pointed to EPC jigs, electronic power controllers, as the most likely first addition to the system after this project. This chapter will give a description of how the current system could be expanded to include this.

The entry point to this would be the button already made in the main window. If the software is expanded to include every part of the test configuration at DSS Horten, then a redesign of the main window should be considered.

Figure 6-1 shows an overview of the current projects in the software, with an example of file structure in the relay matrix project. To start the development of epc configuration, a new project should be made and added to the solution. This project should be loadable from the MainWindow, and it should be able to include functionality from SharedLibraries.

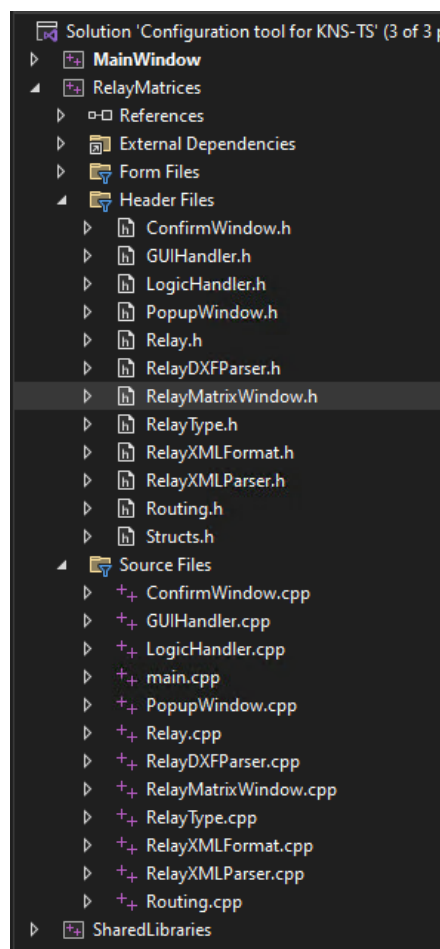


Figure 6-1 Projects in the software.

6 Future development

The development should start by designing a QT user interface. This can be done using the VS Tools plugin, or without any design tools. After the design is done, the functionality needed should be added into the epc project.

The epc project should contain both the user interface layer and the logic layer. This separation should be clear to avoid confusing dependencies.

The user interface layer should contain the code for user interface elements and the manipulation of these. To make this for the epc configuration that is currently in used by the other software in DSS Horten, an analysis of the current XML files needs to be done. This should identify the necessary elements for a user interface. The epc window should have a import and export functionality separately from the relay matrix window.

Validation of data, and the storage of current data should be done in the logic layer. Any manipulation of data should be done in this layer, and the user interface should show the current data in the logic layer. This layer should also include any formatting and parsing of XML or DXF data.

The data layer is in the SharedLibraries. This project should already have the needed functionality to both write and read XML and DXF data. If any additional files type support is needed, this should be added into this project.

7 Discussion

The software created is a good addition to the test system in use at DSS Horten. Discussions with member of the potential future user group at the company showed a promising work procedure from design sketch to DXF file, via the software, to a valid XML configuration file. This would save time and money, for the company in the daily operations.

The software ended up with a well-received UI. Some small changes to the UI were made late in the development to include extra text fields. These fields contain names for the entire matrix. This was not discovered in the original analysis but found later when XML files were compared to test files. The implementation of these changes showed a software that could be modified easily without requiring large changes in other places of the code. This is a good sign for further development and maintenance of the system.

The customer was satisfied with an adaption made to separate the solution into project packages. This would allow additional functionality to be naturally added as new projects for future work.

The code could be better documented, and a tool for documentation, like DOxygen used in other software at the company, could be used. This would make it easier, especially for people without knowledge of the system, to maintain and expand the software.

During testing of the system, concerns were raised by a colleague experienced in the manual configuration of XML files. They were questioning if the new software solution would be quicker than the manual approach. No specific part of the software was considered slow, or ineffective, but it's hard to compete on speed with copy and paste from previous configurations. Even though the import of XML can be done by the software as well, it requires a few extra steps.

It was considered a legitimate concern, and certainly something to keep in mind while further developing the system. The automatic generation of relays from design files were considered a good improvement to the previous solution.

A more thorough test of the system could have been completed. The system is mainly tested by the developer. It's easy to be blind of your own mistakes, and a new user can discover error the developer did not consider.

The overall impression of the system is that it is well functioning. There are no obvious errors found so far, but it is never possible to exclude the possibility of bugs or errors.

8 Conclusion

The final software matches well up to the requirements. The 4 use cases identified from the requirements are all fully implemented with no lacking features. The overall design of the system is well structured with clear separation between the UI layer, the logic layer, and the data layer. All communication in between the layers is handled by the entry points named handlers. The elements needed to create and manipulate a configuration file is implemented, and the resulting export of XML file is on the format currently in use by the test system, KNS-TS.

The software is ready for future expansions. The relay matrix configuration is fully implemented and ready to be taken into use.

8.1 Possible improvements

Some code currently located in the relay matrix project could be moved to the SharedLibraries. Exactly which code to place here is difficult to know until further development has started. Any duplicate code needed for new modules, should optimally be refactored, and moved to SharedLibraries. An example of likely code to move would be the popupWindow, showing a message to the user and closing upon a button press. This is currently in the RelayMatrices project but could be useful in additional modules.

I would also suggest splitting every current project into a separate UI layer project and a logic layer project. This would improve the structure of the code and help enforce the design principles the software was created on.

The development process could have been more effective in the elaboration phase. Setting up the development environment was shown to be more time consuming than anticipated, delaying the first iteration by a week. The time was made up by iteration 2 and 3 having quite a lot of overlapping functionality. Allowing for more effective development of these iterations.

References

- [1] N.O Skeie, «Lecture notes for object-oriented analysis, design, and programming using UML and C#»
- [2] «Extensible Markup Language (XML)». Opened: 3. March 2024. [Online]. Available on: <https://www.w3.org/XML/>
- [3] Q. Liu, «A study of extensible Markup Language (XML)».
- [4] «Extensible Markup Language (XML) 1.0 (Fifth Edition)». Opened: 4. March 2024. [Online]. Available on: <https://www.w3.org/TR/xml/>
- [5] «Extensible Markup Language (XML) 1.1 (Second Edition)». Opened: 4. March 2024. [Online]. Available on: <https://www.w3.org/TR/xml11/#sec-xml11>
- [6] «What is XML (Extensible Markup Language)?», WhatIs. Opened: 3. March 2024. [Online]. Available on: <https://www.techtarget.com/whatis/definition/XML-Extensible-Markup-Language>
- [7] S. Miller, «What Is XML Used For?», Codecademy Blog. Opened: 3. May 2024. [Online]. Available on: <https://www.codecademy.com/resources/blog/what-is-xml-used-for/>
- [8] «Qt VS Tools». Opened: 10. May 2024. [Online]. Available on: <https://doc.qt.io/qtvstools/index.html>
- [9] «Home · Wiki · GNOME / libxml2 · GitLab», GitLab. Opened: 13. May 2024. [Online]. Available on: <https://gitlab.gnome.org/GNOME/libxml2/-/wikis/home>
- [10] kwikius, «kwikius/dxflib». 10. mars 2024. Opened: 14. May 2024. [Online]. Available on: <https://github.com/kwikius/dxflib>

Appendices

Appendix A Task description

Appendix B FURPS+ Analysis

Appendix C FDUCDs

Appendix D SSDs

Appendix E Test plan

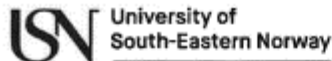
Appendix F Example UI form file

Appendix G Example header file

Appendix H Example cpp file

Appendix I Example XML configuration file

Appendix A Task description



University of
South-Eastern Norway
Faculty of Technology, Natural Sciences and Maritime Sciences, Campus Porsgrunn

FMH606 Master's Thesis

Title: Development of configuration software for a test rig of electrical systems

USN supervisor: Nils-Olav Skeie, Co-supervisor: Leila Ben Saad

External partner: Ove Lehto Pedersen, Kongsberg Defence and Aerospace

Task background:

Kongsberg Defence and Aerospace (KDA) is a Norwegian company producing equipment for the defence and space industry. One of KDA's divisions, Space and Surveillance (DSS), has a branch in Horten. This branch sells electric equipment, like filters, receivers, transmitters, or smaller components, for use in space. Space equipment can't be replaced or repaired once launched, which means there's extreme requirements regarding quality assurance for these products.

A large part of the tests for these products, is automated using inhouse software. This software has already been developed over the last 10-15 years and is constantly being developed further. DSS is looking for a better way to handle configurations, like jigs, relay matrices and wiring in software. These configurations are unique and needs to be customized multiple times for different tests on the same unit. The software uses XML files to customize every unique test setup. DSS would like to simplify this customization by creating a new software that can generate these XML files. This software should make the configuration of new setups a lot quicker and reduce time costly human errors. It would also lower the knowledge needed to setup the configuration, which would free up human resources.

Task description:

- (1) Literature review on XML standards and structures.
- (2) Make the requirement for the configuration tool.
- (3) Give an overview of supported XML methods for the IDE used at KDA and how these can be used for the configuration tool.
- (4) Design the configuration tool.
- (5) Implement configuration of relay matrices in the configuration tool
- (6) Create a test plan and perform a test of the configuration tool.
- (7) Describe how to implement a new module for electronic power control jigs in the configuration tool.

Student category:

IIA, reserved for Industry master (IM) student Krister Pedersen.

Is the task suitable for online students (not present at the campus)? No

Practical arrangements:

IM student working in the company.

Supervision:

As a general rule, the student is entitled to 15-20 hours of supervision. This includes necessary time for the supervisor to prepare for supervision meetings (reading material to be discussed, etc).

Signatures:

Supervisor (date and signature):

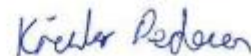
12-FEB-24



Student (write clearly in all capitalized letters): Krister Pedersen

Student (date and signature):

07.02.2024



APPENDIX B

Furps+ Analysis

Functional:

- Add, edit and delete RF relay matrix relays from the configuration.
- Add, edit and delete RF relay matrix routings from the configuration.
- Create a valid XML configuration file.
- Import configuration from XML file.
- Import configuration from DXF file.

Usability:

- GUI should provide available relay matrix relay options.
- Navigation using standard QT elements.
- System should be documented on internal systems using Confluence.
- XML files should be used for output and input.
- DXF files should be used as input.
- Configuration should be stored in a xml file.

Reliability:

- The system should create valid xml files on the format other DSS software requires.
- Files with invalid input should return a message clearly stating where and what is wrong.

Performance:

- System should be able to read and load an XML file under 1000 lines within 10 seconds.

Supportability:

- The software should support the development of additional functionalities.
- System should run as its own .exe file.

+challenges/limitations:

- The system should be developed in C++ 2020.
- The system should be object oriented.
- The system should be using the latest Long-Term-Support version of QT as GUI library.
- The XML output should be version 1.0 using UTF-8 encoding.

Appendix C FDUCDs

Iteration 1

Use case name: Manual input of configuration into the system.

Scope: Configuration Tool

Level: User goal

Primary actor: User

Stakeholders: Owner

Preconditions: Software started. Relay matrix window opened.

Success guarantee: Add data, update data and delete data from user interface.

Main success scenario:

1. Get combo box data from XML file.
2. Add data using text fields and combo boxes.
3. Press “add” button.
4. Confirm valid data.
5. Add data to table.
6. Update GUI with new data.

Extensions:

Update button:

2a Select data from table.

2b Data for selected data shown in GUI.

2c Change 1 or more fields in GUI.

2d Press “update” button.

2e Confirm valid data.

2f Go to step 6.

Delete button:

2a Select data from table.

2b Data for selected data shown in GUI.

2c Press “delete” button.

2d Remove data from table.

2e Go to step 6

Data checks:

4a, 11a Invalid data. (Missing data, invalid values, duplicate data)

4b, 11b Show error message.

4c Go to step 2

Special requirements: none

Technology list: QT GUI

Frequency of occurrence:

Miscellaneous: Data validation done on button press.

Iteration 2

Use case name: Export valid XML configuration file.

Scope: Configuration Tool

Level: User goal

Primary actor: User

Stakeholders: Owner

Preconditions: Software started. Relay matrix window opened.

Success guarantee: Create a valid XML configuration file with data from the system.

Main success scenario:

8. Data imported/input into system.
9. Add valid path into the output path field.
10. Press the Export button.
11. Convert relay and routing data to XML data.
12. Create the XML file at chosen path.
13. Confirm if export was successful.
14. Tell user of successful/unsuccessful export.

Extensions:

3a Invalid path.

3b Tell user of invalid path.

3c Go to step 2.

4a Invalid data.

4b Tell user of invalid data.

4c Go to step 1

Special requirements: none

Technology list: QT GUI, XML library

Frequency of occurrence:

Miscellaneous

Iteration 3

Use case name: Import XML configuration file into the software.

Scope: Configuration Tool

Level: User goal

Primary actor: User

Stakeholders: Owner

Preconditions: Software started. Relay matrix window opened. XML configuration file exists.

Success guarantee: Import data from a valid XML configuration file.

Main success scenario:

6. Add valid path into the import path field.
7. Press the import button.
8. Read XML file.
9. Convert XML data to software format.
10. Show imported data to the user.

Extensions:

2a Invalid path.

2b Tell user of invalid path.

2c Go to step 1.

3a Invalid file type

3b Tell user of error.

3c Go to step 1.

5a Wrong data format/missing data

5b Tell user of invalid data.

5c Go to step 1.

Special requirements: none

Technology list: QT GUI, XML library

Frequency of occurrence:

Miscellaneous

Iteration 4

Use case name: Import DXF file into the software.

Scope: Configuration Tool

Level: User goal

Primary actor: User

Stakeholders: Owner

Preconditions: Software started. Relay matrix window opened. DXF file exists.

Success guarantee: Import data from a valid DXF file.

Main success scenario:

6. Add valid path into the import path field.
7. Press the import button.
8. Read DXF file.
9. Convert DXF data to software format.
10. Show imported data to the user.

Extensions:

2a Invalid path.

2b Tell user of invalid path.

2c Go to step 1.

3a Invalid file type.

3b Tell user of error.

3c Go to step 1.

5a Wrong data format/missing data

5b Tell user of invalid data.

5c Go to step 1.

Special requirements: none

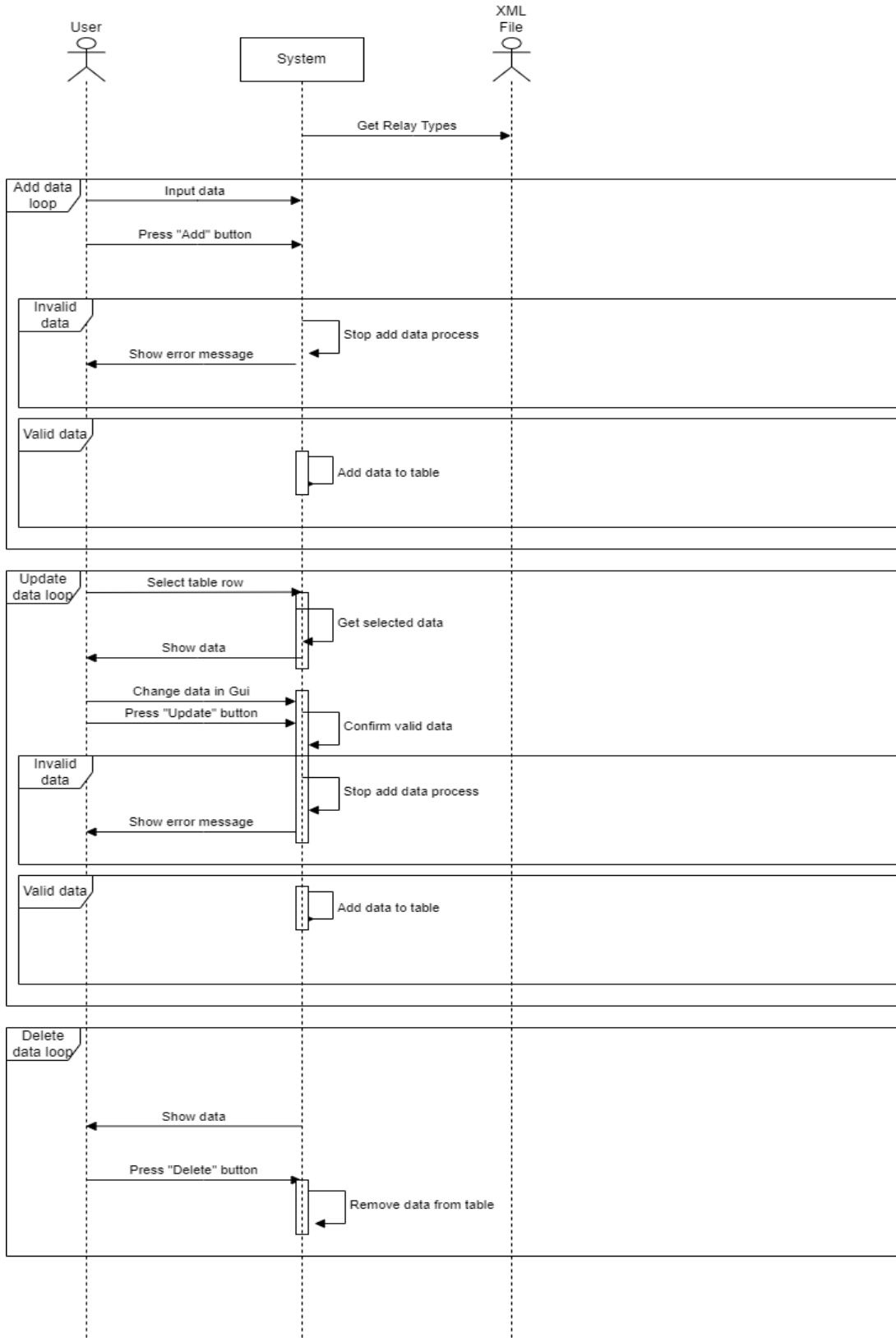
Technology list: QT GUI, XML library

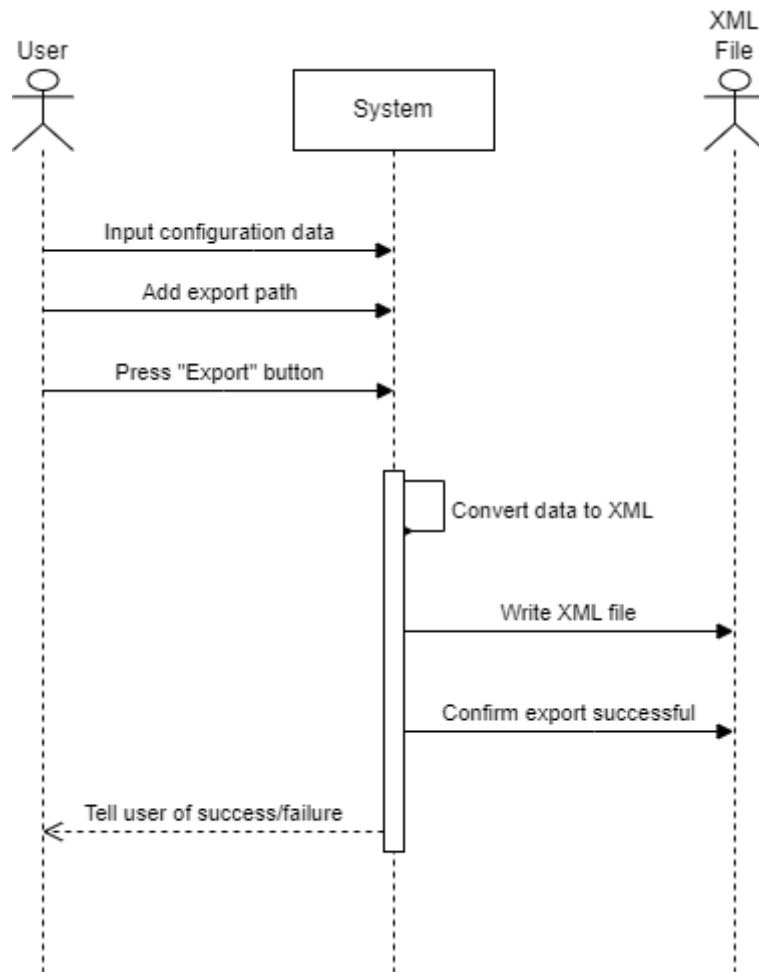
Frequency of occurrence:

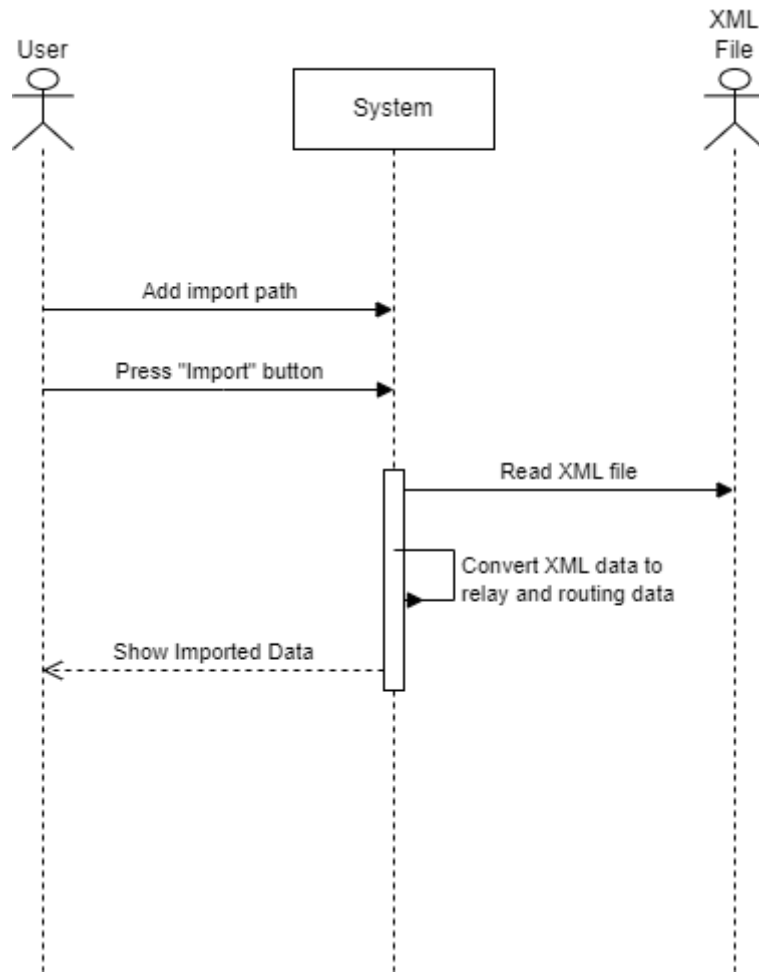
Miscellaneous

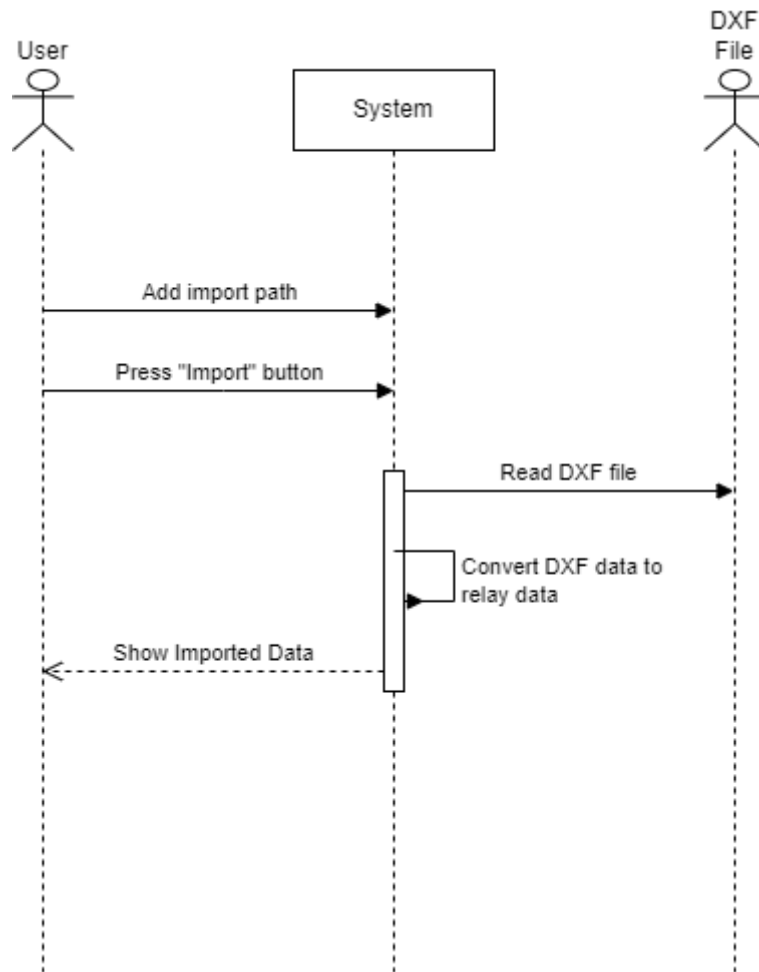
Appendix D SSDs

Iteration 1









Appendix E Test plan

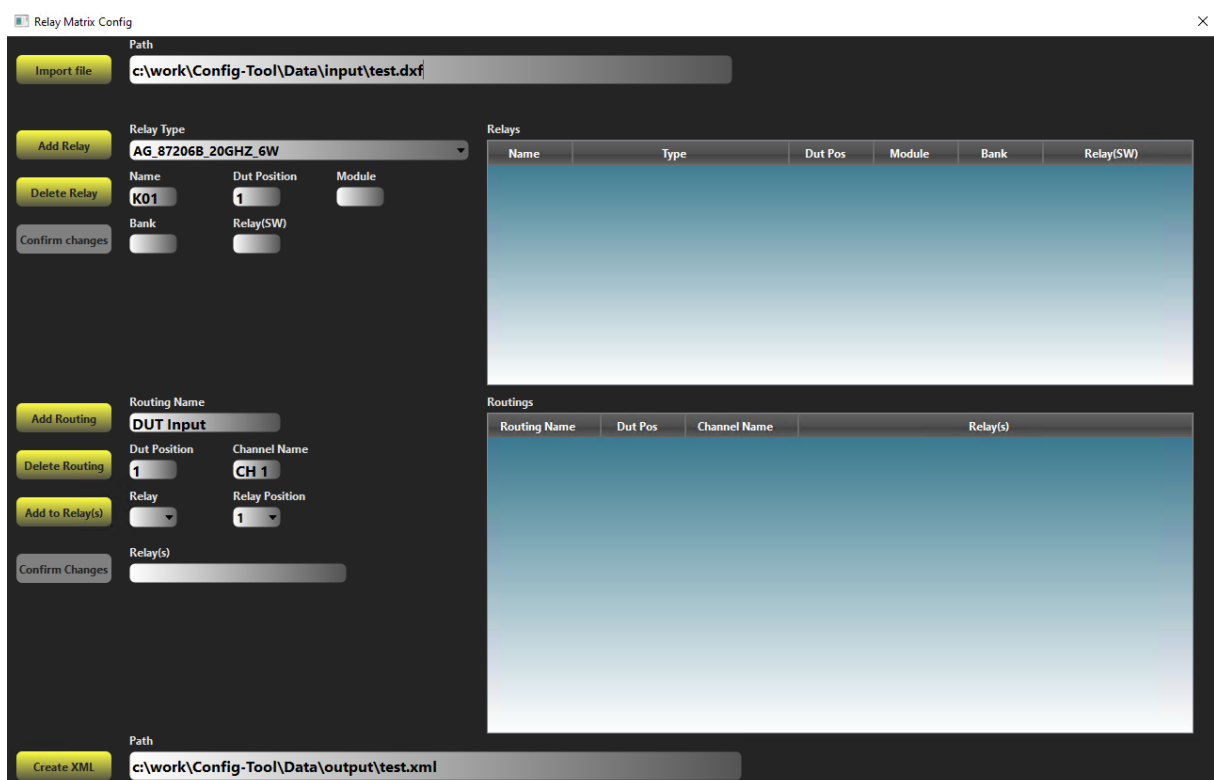
Introduction

This test plan describes the testing done after developing the relay matrix module of the KNS-TS configuration tool. It includes a short system description with a user interface screenshot and a short explanation of the different functionality in the user interface.

System description

The software being tested is the KNS-TS configuration tool. The software is used to import, change, and export configuration data. This data is used by the KNS-TS software to setup for hardware testing. This test plan describes the testing of the relay matrix functionality of the system.

The configuration tool can be used to import design files, DXF files, or configuration files, XML files, or a new configuration can be created. Data can be added, edited, or removed using the graphical user interface. The completed configuration data can be exported as an XML file, ready for use in KNS-TS. A screenshot of the user interface can be seen below.



To import data, find the correct path, with correct file extension, and input it into the path field. When you click import file, the file in the path field will be read and parsed into system data. The data read will be shown in the tables. When importing data, all current data will be replaced. A confirmation window will appear to alert the user of this.

The software is separated into Relays and Routings. Relay data describes the physical relays and their connections in the test system. Routing data describes saved routings in the matrix. Both of these have add, delete and change functionality in the user interface. The currently saved data is visible in the table fields on the right side.

The “Create XML” button creates a new XML file at the chosen path. This will overwrite any existing file with the same path. This file is ready for use in the KNS-TS software.

Test setup

To test the import functionality of the system, a DXF file and an XML file are required. In the test matrix the dxf file is used for setting up the test environment, giving the tester a set of starting relays. The XML file is only used to generate an alternative configuration. Either one of these files could have been used as a starting point.

The dxf file requires the KNS-TS naming convention to be followed accurately. This naming convention requires 2 important text strings for each relay. Firstly, the name of the Relay is used to add information of the relay position in the system. The name should be on the form “Relayname_M(Module)_B(Bank)_S(Switch)”. An example would be: “K01_M1_B1_S1”. In addition to the name, the type of the Relay must be available in the dxf file. The type is on the form of “Relayname_type”. An example of this would be: “K01_87206B”. If Altium Designer is used to make the dxf files, these strings must be visible on the graphical design file. An example of the correct dxf output can be seen in Figure 8-1. The dxf file will only import relay information, as no routing information is available in the design file.

```

140918 1.351063829787234
140919 1
140920 K01_87104B
140921 0
140922 TEXT
140923 5
140924 75E
140925 8
140926 SHEET
140927 10
140928 165.8070177430802
140929 20
140930 243.731914893617
140931 30
140932 0.0
140933 40
140934 1.351063829787234
140935 1
140936 K01_M1_B3_S1
140937 0
140938 POLYLINE
140939 5
140940 75F
140941 8
140942 SHEET
140943 66
    
```

Figure 8-1 A few lines of the dxf file. It contains the needed information for relay K01

The XML file requires specific tags used by the KNS-TS system, to be able to read and convert the XML file into data for the configuration tool. The tags can be seen in Figure 8-2. The figure has the needed data for 1 relay and 1 routing.

364	□	<relay>
365		<name>K10</name>
366		<type>HP_8765D_40_GHZ_2W</type>
367	□	<controller>
368		<functional_name>F_Relay_Controller</functional_name>
369		<dut_pos>10</dut_pos>
370		<module>10</module>
371		<bank>10</bank>
372		<relaysw>10</relaysw>
373		</controller>
374	□	<relay_pos>
375		<name>K10_1</name>
376		<relay_position>1</relay_position>
377		</relay_pos>
378	□	<relay_pos>
379		<name>K10_2</name>
380		<relay_position>2</relay_position>
381		</relay_pos>
382		</relay>
383	□	<routing>
384		<functional_name>CH 1</functional_name>
385	□	<config>
386		<name>Dut Input</name>
387		<relay_pos>K01_1, K02_1</relay_pos>
388		</config>
389		</routing>

Figure 8-2 Example xml file

Test tasks

The set of tasks and expected results are presented in Figure 8-3. The red tasks are expected errors.

Description test case	Expected result
Start software and open relay matrix window	Relay matrix GUI shown
Import dxf: Import test file c:\work\Config-Tool\Data\input\test.dxf	A set of relays imported into system
Add Relay: Name K05, Type AG_87206B_20GHZ_6W, DutPos 1, Module 1, Bank 1, Relay(SW) 1	Relay added to Relay table. No error
Add Relay: Name K06, Type AG_87204B_20GHZ_4W, DutPos 1, Module 1, Bank 1, Relay(SW) 2	Relay added to Relay table. No error
Add Relay: Name K07, AG_N1810UL_26_5GHZ_2W, DutPos 1, Module 1, Bank 2, Relay(SW) 1	Relay added to Relay table. No error
Error message: Try adding another Relay with name K01	Relay NOT added to Relay table. Error message shown
Error message: Try adding Relay with empty field(s)	Relay NOT added to Relay table. Error message shown
Error message: Try adding Relay with illegal values	Relay NOT added to Relay table. Error message shown
Change Name on relay K07 to K08	Relay updated in Relay table. No error
Change data of K08 to TypeHP_8765B_20_GHZ_2W, DutPos 2, Module 2, Bank 2, Relay(SW) 2	Relay updated in Relay table. No error
Delete Relay K08	Relay removed from Relay table. No error
Add Routing: Name DUT Input, DutPos 1, ChannelName CH 1, Relay(s) K01_1, K02_1	Routing added to Routing table. No error
Add Routing: Name DUT Output, DutPos 1, ChannelName CH 2, Relay(s) K01_2, K02_2	Routing added to Routing table. No error
Add Routing: Name Temp, DutPos 1, ChannelName CH 2, Relay(s) K01_1	Routing added to Routing table. No error
Change routing Temp: add relay k02, position 2 to relay(s) string using comboboxes and "add to relay(s)" button	Routing updated in Routing table. No error
Error message: Try adding Routing with empty field(s)	Routing NOT added to Routing table. Error message shown
Error message: Try adding Routing with illegal values	Routing NOT added to Routing table. Error message shown
Delete routing Temp	Routing removed from Routing table. No error
Create XML: Export xml config to a path of your choosing. Call the file test.xml	xml file created. Result label shown
Import XML: Import test file c:\work\Config-Tool\Data\input\test.xml	Previous data replaced with new data

Figure 8-3 Set of tasks and expected results

Appendix F Example UI form file snippet

```

1510 <widget class="QLineEdit" name="exportPathEdit">
1511   <property name="geometry">
1512     <rect>
1513       <x>130</x>
1514       <y>760</y>
1515       <width>651</width>
1516       <height>31</height>
1517     </rect>
1518   </property>
1519   <property name="text">
1520     <string>c:\work\Config-Tool\Data\output\test.xml</string>
1521   </property>
1522 </widget>
1523 <widget class="QLabel" name="label_6">
1524   <property name="geometry">
1525     <rect>
1526       <x>130</x>
1527       <y>740</y>
1528       <width>49</width>
1529       <height>16</height>
1530     </rect>
1531   </property>
1532   <property name="text">
1533     <string>Path</string>
1534   </property>
1535 </widget>
1536 <widget class="QTableWidget" name="relayTable">
1537   <property name="geometry">
1538     <rect>
1539       <x>510</x>
1540       <y>110</y>
1541       <width>751</width>
1542       <height>261</height>
1543     </rect>
1544   </property>
1545   <property name="selectionMode">
1546     <enum>QAbstractItemView::SingleSelection</enum>
1547   </property>
1548   <property name="selectionBehavior">
1549     <enum>QAbstractItemView::SelectRows</enum>
1550   </property>
1551   <property name="cornerButtonEnabled">
1552     <bool>false</bool>
1553   </property>
1554   <attribute name="horizontalHeaderCascadingSectionResizes">
1555     <bool>false</bool>
1556   </attribute>
1557   <attribute name="horizontalHeaderDefaultSectionSize">
1558     <number>90</number>

```


Appendix G Example header file

RelayMatrixWindow.h

```
#pragma once

#include <QtWidgets/QDialog>
#include "ui_RelayMatrixWindow.h"
#include "PopupWindow.h"
#include "GUIHandler.h"
#include "Structs.h"

class RelayMatrixWindow : public QDialog
{
    Q_OBJECT
public:
    __declspec(dllexport) RelayMatrixWindow(QWidget *parent = nullptr);
    ~RelayMatrixWindow() {};

protected:

private:
    Ui::relayMatrixWindowClass ui;
    //Ui::PopupWindow popupWindowUi;
    void ConnectSlots();
    void UpdateTables();
    void GetRelayTypes();

    GUIHandler* uiHandler;
    PopupWindow* popupWindow; /*
    ConfirmWindow* confirmWindow;*/

public slots:
    void RelayDataChangedSlot(const QString&);
    void TypeComboboxChangedSlot(int i);
    void relayComboboxChangedSlot(int i);

    void RoutingDataChangedSlot(const QString&);
    void SelectedRelaySlot();
    void SelectedRoutingSlot();

    void ImportConfigSlot();
    void ExportConfigSlot();

    void AddRelaySlot();
    void DeleteRelaySlot();
    void UpdateRelaySlot();

    void AddRoutingSlot();
    void DeleteRoutingSlot();
    void UpdateRoutingSlot();
    void AddRelayRoutingSlot();

signals:
};
```

Appendix H Example cpp file

```
RelayMatrixWindow.cpp
```

```
#include "RelayMatrixWindow.h"
```

```
#include "qmessagebox.h"
```

```
RelayMatrixWindow::RelayMatrixWindow(QWidget *parent)
```

```
    : QDialog(parent)
```

```
{
```

```
    ui.setupUi(this);
```

```
    uiHandler = GUIHandler::GetInstance();
```

```
    // Connect Buttons
```

```
    ConnectSlots();
```

```
    ui.exportLabel->setVisible(false);
```

```
    //Table customisation
```

```
    ui.relayTable->setColumnWidth(1, 230);
```

```
    ui.routingTable->setColumnWidth(0, 120);
```

```
    ui.routingTable->setColumnWidth(2, 120);
```

```
    ui.relayTable->setEditTriggers(QAbstractItemView::NoEditTriggers);
```

```
    ui.routingTable->setEditTriggers(QAbstractItemView::NoEditTriggers);
```

```
    GetRelayTypes();
```

```
}
```

```
//Setup event handlers
```

```
void
```

```
RelayMatrixWindow::ConnectSlots()
```

```
{
```

```
    connect(ui.relayTable, SIGNAL(itemSelectionChanged()), this,  
           SLOT(SelectedRelaySlot()));
```

```
    connect(ui.routingTable, SIGNAL(itemSelectionChanged()), this,  
           SLOT(SelectedRoutingSlot()));
```

Appendices

```
connect(ui.importButton, SIGNAL(clicked()), this, SLOT(ImportConfigSlot()));
connect(ui.exportButton, SIGNAL(clicked()), this, SLOT(ExportConfigSlot()));

connect(ui.addRelayButton, SIGNAL(clicked()), this, SLOT(AddRelaySlot()));
connect(ui.deleteRelayButton, SIGNAL(clicked()), this,
SLOT(DeleteRelaySlot()));
connect(ui.changeRelayButton, SIGNAL(clicked()), this,
SLOT(UpdateRelaySlot()));

connect(ui.addRoutingButton, SIGNAL(clicked()), this,
SLOT(AddRoutingSlot()));
connect(ui.deleteRoutingButton, SIGNAL(clicked()), this,
SLOT(DeleteRoutingSlot()));
connect(ui.addRelaytoRoutingButton, SIGNAL(clicked()), this,
SLOT(AddRelayRoutingSlot()));
connect(ui.changeRoutingButton, SIGNAL(clicked()), this,
SLOT(UpdateRoutingSlot()));

connect(ui.relayNameEdit, SIGNAL(textEdited(const QString&)), this,
SLOT(RelayDataChangedSlot(const QString&)));
connect(ui.relayDutPosEdit, SIGNAL(textEdited(const QString&)), this,
SLOT(RelayDataChangedSlot(const QString&)));
connect(ui.relayBankEdit, SIGNAL(textEdited(const QString&)), this,
SLOT(RelayDataChangedSlot(const QString&)));
connect(ui.relayModuleEdit, SIGNAL(textEdited(const QString&)), this,
SLOT(RelayDataChangedSlot(const QString&)));
connect(ui.relaySWEdit, SIGNAL(textEdited(const QString&)), this,
SLOT(RelayDataChangedSlot(const QString&)));
connect(ui.relayTypeComboBox, SIGNAL(currentIndexChanged(int)), this,
SLOT(TypeComboboxChangedSlot(int)));
connect(ui.relayComboBox, SIGNAL(currentIndexChanged(int)), this,
SLOT(relayComboboxChangedSlot(int)));

connect(ui.routingNameEdit, SIGNAL(textEdited(const QString&)), this,
SLOT(RoutingDataChangedSlot(const QString&)));
connect(ui.routingDutPosEdit, SIGNAL(textEdited(const QString&)), this,
SLOT(RoutingDataChangedSlot(const QString&)));
connect(ui.nameEdit, SIGNAL(textEdited(const QString&)), this,
SLOT(RoutingDataChangedSlot(const QString&)));
connect(ui.relayRoutingEdit, SIGNAL(textEdited(const QString&)), this,
SLOT(RoutingDataChangedSlot(const QString&)));
}

//Get relay type data from logic layer
```

```

void
RelayMatrixWindow::GetRelayTypes()
{
    std::vector<std::string> typeVector = uiHandler->GetRelayTypes();
    for (std::string type : typeVector)
    {
        ui.relayTypeComboBox->addItem(QString::fromStdString(type));
    }
}

//Get table data from logic layer
void
RelayMatrixWindow::UpdateTables()
{
    std::pair < std::vector<std::vector<std::string>>,
std::vector<std::vector<std::string>>> data = uiHandler->GetData();

    //clear old data
    while (ui.relayTable->rowCount() > 0)
    {
        ui.relayTable->removeRow(ui.relayTable->rowCount() - 1);
    }
    while (ui.routingTable->rowCount() > 0)
    {
        ui.routingTable->removeRow(ui.routingTable->rowCount() - 1);
    }
    ui.relayComboBox->clear();

    for (std::vector<std::string> rowData : data.first)
    {
        int rowCount = ui.relayTable->rowCount();
        int columnCount = ui.relayTable->columnCount();

        ui.relayTable->insertRow(rowCount);

        QTableWidgetItem* name = new QTableWidgetItem();
        QTableWidgetItem* type = new QTableWidgetItem();
        QTableWidgetItem* dutPos = new QTableWidgetItem();
        QTableWidgetItem* module = new QTableWidgetItem();
    }
}

```

```

QTableWidgetItem* bank = new QTableWidgetItem();
QTableWidgetItem* relaySW = new QTableWidgetItem();

name->setText(QString::fromStdString(rowData.at(0)));
type->setText(QString::fromStdString(rowData.at(1)));
dutPos->setText(QString::fromStdString(rowData.at(2)));
module->setText(QString::fromStdString(rowData.at(3)));
bank->setText(QString::fromStdString(rowData.at(4)));
relaySW->setText(QString::fromStdString(rowData.at(5)));

std::vector<QTableWidgetItem*> fields = { name, type, dutPos, module,
bank, relaySW };

for (size_t i = 0; i < columnCount; i++)
{
    ui.relayTable->setItem(rowCount, i, fields.at(i));
}
ui.relayComboBox->addItem(name->text());
}

//ui.routingTable->clear();

for (std::vector<std::string> rowData : data.second)
{
    int rowCount = ui.routingTable->rowCount();
    int columnCount = ui.routingTable->columnCount();

    ui.routingTable->insertRow(rowCount);

    QTableWidgetItem* name = new QTableWidgetItem();
    QTableWidgetItem* dutPos = new QTableWidgetItem();
    QTableWidgetItem* channelName = new QTableWidgetItem();
    QTableWidgetItem* relayRouting = new QTableWidgetItem();

    name->setText(QString::fromStdString(rowData.at(0)));
    dutPos->setText(QString::fromStdString(rowData.at(1)));
    channelName->setText(QString::fromStdString(rowData.at(2)));
    relayRouting->setText(QString::fromStdString(rowData.at(3)));
}

```

```

        std::vector<QTableWidgetItem*> fields = { name, dutPos, channelName,
        relayRouting };

        for (size_t i = 0; i < columnCount; i++)
        {
            ui.routingTable->setItem(rowCount, i, fields.at(i));
        }

    }

    std::vector<std::string> names = uiHandler->GetMatrixNames();
    if (!names.at(0).empty())
    {
        ui.functionalNameEdit->setText(QString::fromStdString(names.at(0)));
    }
    if (!names.at(1).empty())
    {
        ui.logicalNameEdit->setText(QString::fromStdString(names.at(1)));
    }
}

//Handle select row in relay table
void
RelayMatrixWindow::SelectedRelaySlot()
{
    QList<QTableWidgetItem*> items = ui.relayTable->selectedItems();
    if (items.count() == 0)
    {
        return;
    }
    QString name = items.at(0)->text();
    ui.relayNameEdit->setText(name);

    //Find combobox index from string
    for (auto i = 0; i < ui.relayTypeComboBox->count(); i++)
    {
        QString comboBoxText = ui.relayTypeComboBox->itemText(i);
        QString type = items.at(1)->text();
        if (comboBoxText == type)
        {

```

```

        ui.relayTypeComboBox->setCurrentIndex(i);
        break;
    }
}

QString dutPos = items.at(2)->text();
ui.relayDutPosEdit->setText(dutPos);

QString module = items.at(3)->text();
ui.relayModuleEdit->setText(module);

QString bank = items.at(4)->text();
ui.relayBankEdit->setText(bank);

QString relaySW = items.at(5)->text();
ui.relaySWEdit->setText(relaySW);

ui.changeRelayButton->setDisabled(true);
}

//Handle select row in routing table
void
RelayMatrixWindow::SelectedRoutingSlot()
{
    QList<QTableWidgetItem*> items = ui.routingTable->selectedItems();

    if (items.count() == 0)
    {
        return;
    }
    QString name = items.at(0)->text();
    ui.routingNameEdit->setText(name);

    QString dutPos = items.at(1)->text();
    ui.routingDutPosEdit->setText(dutPos);

    QString channelName = items.at(2)->text();
    ui.nameEdit->setText(channelName);
}

```

```

QString relayRouting = items.at(3)->text();
ui.relayRoutingEdit->setText(relayRouting);

ui.changeRoutingButton->setDisabled(true);
}

//Handle changed relay data editfield action
void
RelayMatrixWindow::RelayDataChangedSlot(const QString&)
{
    if (ui.relayTable->selectedItems().count() > 0)
    {
        ui.changeRelayButton->setEnabled(true);
    }
}

//Handle change routing data action
void
RelayMatrixWindow::RoutingDataChangedSlot(const QString& text)
{
    if (ui.routingTable->selectedItems().count() > 0)
    {
        ui.changeRoutingButton->setEnabled(true);
    }
}

//Handle change type combobox value
void
RelayMatrixWindow::TypeComboboxChangedSlot(int i)
{
    if (ui.relayTable->selectedItems().count() > 0)
    {
        ui.changeRelayButton->setEnabled(true);
    }
}

void RelayMatrixWindow::relayComboboxChangedSlot(int i)
{
    int rowCount = ui.relayTable->rowCount();

```



```

std::string name = ui.relayComboBox->currentText().toStdString();
std::string type;
for (size_t i = 0; i < rowCount; i++)
{
    if (name == ui.relayTable->item(i,0)->text().toStdString())
    {
        type = ui.relayTable->item(i, 1)->text().toStdString();
        break;
    }
}
ui.relayPosComboBox->clear();
for (int i : uiHandler->GetPositions(type))
{
    ui.relayPosComboBox->addItem(QString::fromStdString(std::to_string(i)));
}
}

//Handle import data action
void
RelayMatrixWindow::ImportConfigSlot()
{
    std::string path = ui.importEdit->text().toStdString();
    int import = 0;

    QMessageBox msg;
    msg.setText("Importing data replaces all previous data in the software. \nAre
you sure you want to continue?");
    msg.setStandardButtons(QMessageBox::Ok | QMessageBox::Cancel);
    msg.setWindowTitle("Confirm action");
    msg.setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);
    //msg.setStyleSheet("QPushButton{color: E3E3E3; background-color:
qlineargradient(x1 : 0, y1 : 0, x2 : 0, y2 : 1, stop : 0 #ffff44, stop:1
#545444);} QLabel{color: #E3E3E3; font: bold 12px;} QMessageBox{background-color:
#242424; }");

    ValidationCheck check;
    int ret = msg.exec();
    switch (ret) {
    case QMessageBox::Ok:
        check = uiHandler->ImportData(path);

```

```

if (check.isValid == true)
{
    UpdateTables();
    if (!check.errorMessage.empty())
    {
        popupWindow = new PopupWindow(this);
        popupWindow->SetMessage("Import complete. \nThe following
relay(s) and Routing(s) had errors and were discarded:\n" + check.errorMessage);
        popupWindow->ShowPopup();
    }
}
else
{
    popupWindow = new PopupWindow(this);
    popupWindow->SetMessage("Import failed. Make sure filepath is correct
and includes extension (.xml or .dxf)");
    popupWindow->ShowPopup();
}
break;
case QMessageBox::Cancel:
    return;
    break;
}
}

//Handle export data action
void
RelayMatrixWindow::ExportConfigSlot()
{
    ui.exportLabel->setVisible(false);

    QMessageBox msg;
    msg.setText("Exporting XML file. This will replace any existing file with the
same name at target location. \nAre you sure you want to continue?");
    msg.setStandardButtons(QMessageBox::Ok | QMessageBox::Cancel);
    msg.setWindowTitle("Confirm export");
    msg.setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);

    std::string path = ui.exportPathEdit->text().toStdString();
    std::string logicalName = ui.logicalNameEdit->text().toStdString();

```

```

std::string functionalName = ui.functionalNameEdit->text().toStdString();
int signs = 0;
int ret = msg.exec();

switch (ret) {
case QMessageBox::Ok:

    signs = uiHandler->ExportData(path, logicalName, functionalName);

    if (signs < 100)
    {
        popupWindow = new PopupWindow(this);
        popupWindow->SetMessage("XML export error. Not enough data to
print.");
        popupWindow->ShowPopup();
    }
    else
    {
        ui.exportLabel->setVisible(true);
        ui.exportLabel->setText(QString::fromStdString("XML export printed "
+ std::to_string(signs) + " signs"));
    }
    break;
case QMessageBox::Cancel:
    return;
    break;
}
}

//Handle add relay data action
void
RelayMatrixWindow::AddRelaySlot()
{
    int rowCount = ui.relayTable->rowCount();
    int columnCount = ui.relayTable->columnCount();

    std::string nameString = ui.relayNameEdit->text().simplified().toStdString();
    std::string typeString = ui.relayTypeComboBox->currentText().toStdString();
    std::string dutPosString = ui.relayDutPosEdit-
>text().simplified().toStdString();

```

```

    std::string moduleString = ui.relayModuleEdit->
text().simplified().toStdString();
    std::string bankString = ui.relayBankEdit->text().simplified().toStdString();
    std::string relaySWString = ui.relaySWEdit->
text().simplified().toStdString();

    std::vector<std::string> stringValues = { nameString, typeString,
dutPosString, moduleString, bankString, relaySWString };

    ValidationCheck check = uiHandler->AddRelay(stringValues);
    if (check.isValid)
    {
        //update gui
        UpdateTables();

    }
    else
    {
        //error message
        popupWindow = new PopupWindow(this);
        popupWindow->SetMessage(check.errorMessage);
        popupWindow->ShowPopup();
    }
    ui.changeRelayButton->setDisabled(true);
}

//Handle delete relay data action
void
RelayMatrixWindow::DeleteRelaySlot()
{
    QList<QTableWidgetItem*> items = ui.relayTable->selectedItems();
    if (items.isEmpty())
    {
        popupWindow = new PopupWindow(this);
        popupWindow->SetMessage("No Relay selected");
        popupWindow->ShowPopup();
        return;
    }

    std::string name = items.at(0)->text().toStdString();

```

```

if (uiHandler->DeleteRelay(name))
{
    int row = items.at(0)->row();
    ui.relayTable->removeRow(row);
    ui.relayComboBox->removeItem(row);
}
ui.changeRelayButton->setDisabled(true);
}

//Handle change relay data action
void
RelayMatrixWindow::UpdateRelaySlot()
{
    QList<QTableWidgetItem*> items = ui.relayTable->selectedItems();
    int row = items.at(0)->row();
    if (items.isEmpty())
    {
        popupWindow = new PopupWindow(this);
        popupWindow->SetMessage("No Relay selected");
        popupWindow->ShowPopup();
        return;
    }

    std::string nameString = ui.relayNameEdit->text().simplified().toStdString();
    std::string typeString = ui.relayTypeComboBox->currentText().toStdString();
    std::string dutPosString = ui.relayDutPosEdit-
>text().simplified().toStdString();
    std::string moduleString = ui.relayModuleEdit-
>text().simplified().toStdString();
    std::string bankString = ui.relayBankEdit->text().simplified().toStdString();
    std::string relaySWString = ui.relaySWEdit-
>text().simplified().toStdString();

    std::vector<std::string> oldData;
    for(QTableWidgetItem* item : items)
    {
        std::string text = item->text().toStdString();
        oldData.emplace_back(text);
    }
}

```

```
std::vector<std::string> newData = { nameString, typeString, dutPosString,
moduleString, bankString, relaySWString };
```

```
ValidationCheck check = uiHandler->UpdateRelay(newData, oldData);
```

```
if (!check.isValid)
```

```
{
```

```
    popupWindow = new PopupWindow(this);
```

```
    popupWindow->SetMessage(check.errorMessage);
```

```
    popupWindow->ShowPopup();
```

```
    return;
```

```
}
```

```
else
```

```
{
```

```
    int columnCount = ui.relayTable->columnCount();
```

```
    for (size_t i = 0; i < items.count(); i++)
```

```
    {
```

```
        items.at(i)->setText(QString::fromStdString(newData.at(i)));
```

```
    }
```

```
    if (newData.at(0) != oldData.at(0))
```

```
    {
```

```
        ui.relayComboBox->setItemText(row,
QString::fromStdString(newData.at(0)));
```

```
    }
```

```
}
```

```
    ui.changeRelayButton->setDisabled(true);
```

```
}
```

```
//Handle add routing data action
```

```
void
```

```
RelayMatrixWindow::AddRoutingSlot()
```

```
{
```

```
    int rowCount = ui.routingTable->rowCount();
```

```
    int columnCount = ui.routingTable->columnCount();
```

```
    std::string nameString = ui.routingNameEdit-
>text().simplified().toStdString();
```

```
    std::string dutPosString = ui.routingDutPosEdit-
>text().simplified().toStdString();
```

```
    std::string channelNameString = ui.nameEdit-
>text().simplified().toStdString();
```

```

    std::string relayRoutingString = ui.relayRoutingEdit-
>text().simplified().toStdString();

    std::vector<std::string> stringValues = { nameString, dutPosString,
channelNameString, relayRoutingString };

    ValidationCheck check = uiHandler->AddRouting(stringValues);

    if (check.isValid)
    {
        ui.routingTable->insertRow(rowCount);

        QTableWidgetItem* name = new QTableWidgetItem();
        QTableWidgetItem* dutPos = new QTableWidgetItem();
        QTableWidgetItem* channelName = new QTableWidgetItem();
        QTableWidgetItem* relayRouting = new QTableWidgetItem();
        name->setText(ui.routingNameEdit->text().simplified());
        dutPos->setText(ui.routingDutPosEdit->text().simplified());
        channelName->setText(ui.nameEdit->text().simplified());
        relayRouting->setText(ui.relayRoutingEdit->text().simplified());

        std::vector<QTableWidgetItem*> fields = { name, dutPos, channelName,
relayRouting };

        for (size_t i = 0; i < columnCount; i++)
        {
            ui.routingTable->setItem(rowCount, i, fields.at(i));
        }
    }
    else
    {
        popupWindow = new PopupWindow(this);
        popupWindow->SetMessage(check.errorMessage);
        popupWindow->ShowPopup();
    }
}

//Handle delete routing data action
void
RelayMatrixWindow::DeleteRoutingSlot()

```

```

{
    QList<QTableWidgetItem*> items = ui.routingTable->selectedItems();
    if (items.isEmpty())
    {
        popupWindow = new PopupWindow(this);
        popupWindow->SetMessage("No Routing selected");
        popupWindow->ShowPopup();
        return;
    }
    int row = items.at(0)->row();
    if (uiHandler->DeleteRouting(row))
    {
        ui.routingTable->removeRow(row);
    }
    ui.changeRoutingButton->setDisabled(true);
}

//Handle change routing data action
void
RelayMatrixWindow::UpdateRoutingSlot()
{
    QList<QTableWidgetItem*> items = ui.routingTable->selectedItems();
    if (items.isEmpty())
    {
        popupWindow = new PopupWindow(this);
        popupWindow->SetMessage("No Relay selected");
        popupWindow->ShowPopup();
        return;
    }

    std::string nameString = ui.routingNameEdit-
>text().simplified().toString();
    std::string dutPosString = ui.routingDutPosEdit-
>text().simplified().toString();
    std::string channelNameString = ui.nameEdit-
>text().simplified().toString();
    std::string relayRoutingString = ui.relayRoutingEdit-
>text().simplified().toString();

    std::vector<std::string> oldData;

```



```

for (QTableWidgetItem* item : items)
{
    std::string text = item->text().simplified().toStdString();
    oldData.emplace_back(text);
}

std::vector<std::string> newData = { nameString, dutPosString,
channelNameString, relayRoutingString };

int row = items.at(0)->row();
ValidationCheck check = uiHandler->UpdateRouting(newData, oldData, row);

if (!check.isValid)
{
    popupWindow = new PopupWindow(this);
    popupWindow->SetMessage(check.errorMessage);
    popupWindow->ShowPopup();
    return;
}
else
{
    int columnCount = ui.routingTable->columnCount();
    for (size_t i = 0; i < items.count(); i++)
    {
        items.at(i)->setText(QString::fromStdString(newData.at(i)));
    }
}
ui.changeRoutingButton->setDisabled(true);
}

//Handle add relayrouting data action
void
RelayMatrixWindow::AddRelayRoutingSlot()
{
    QString relayRouting;
    QString newString;
    relayRouting = ui.relayRoutingEdit->text();

    relayRouting = relayRouting.trimmed();

```

```
if (relayRouting.isEmpty())
{
    relayRouting = ui.relayComboBox->currentText() + "_" +
ui.relayPosComboBox->currentText();
}
else if(relayRouting.endsWith(", "))
{
    newString = " " + ui.relayComboBox->currentText() + "_" +
ui.relayPosComboBox->currentText();
    relayRouting += newString;
}
else
{
    newString = ", " + ui.relayComboBox->currentText() + "_" +
ui.relayPosComboBox->currentText();
    relayRouting += newString;
}
ui.relayRoutingEdit->setText(relayRouting);
ui.changeRoutingButton->setEnabled(true);
}
```

Appendix J Example XML configuration file

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<test_setup>
  <jigs>
    <!-->
    <Relay_matrix>
      <matrix>
        <logical_name>RF_RELAY_MATRIX_1</logical_name>
        <port>

<functional_name>FJ_RF_RELAY_MATRIX_1</functional_name>
        <dut_pos>1</dut_pos>
      </port>
    </matrix>
    <relay>
      <name>K01</name>
      <type>AG_87206C_26_5GHZ_6W</type>
      <controller>

<functional_name>F_Relay_Controller</functional_name>
        <dut_pos>1</dut_pos>
        <module>1</module>
        <bank>1</bank>
        <relaysw>1</relaysw>
      </controller>
      <relay_pos>
        <name>K01_0</name>
        <relay_position>0</relay_position>
      </relay_pos>
      <relay_pos>
        <name>K01_1</name>
        <relay_position>1</relay_position>

```

```

</relay_pos>
<relay_pos>
    <name>K01_2</name>
    <relay_position>2</relay_position>
</relay_pos>
<relay_pos>
    <name>K01_3</name>
    <relay_position>3</relay_position>
</relay_pos>
<relay_pos>
    <name>K01_4</name>
    <relay_position>4</relay_position>
</relay_pos>
<relay_pos>
    <name>K01_5</name>
    <relay_position>5</relay_position>
</relay_pos>
<relay_pos>
    <name>K01_6</name>
    <relay_position>6</relay_position>
</relay_pos>
</relay>
<relay>
    <name>K02</name>
    <type>AG_87206B_20GHZ_6W</type>
    <controller>

<functional_name>F_Relay_Controller</functional_name>
    <dut_pos>2</dut_pos>
    <module>2</module>
    <bank>2</bank>
    <relaysw>2</relaysw>
</controller>
<relay_pos>

```

```

        <name>K02_0</name>
        <relay_position>0</relay_position>
    </relay_pos>
    <relay_pos>
        <name>K02_1</name>
        <relay_position>1</relay_position>
    </relay_pos>
    <relay_pos>
        <name>K02_2</name>
        <relay_position>2</relay_position>
    </relay_pos>
    <relay_pos>
        <name>K02_3</name>
        <relay_position>3</relay_position>
    </relay_pos>
    <relay_pos>
        <name>K02_4</name>
        <relay_position>4</relay_position>
    </relay_pos>
    <relay_pos>
        <name>K02_5</name>
        <relay_position>5</relay_position>
    </relay_pos>
    <relay_pos>
        <name>K02_6</name>
        <relay_position>6</relay_position>
    </relay_pos>
</relay>
<relay>
    <name>K03</name>
    <type>AG_87204B_20GHZ_4W</type>
    <controller>

<functional_name>F_Relay_Controller</functional_name>

```

```
<dut_pos>3</dut_pos>
<module>3</module>
<bank>3</bank>
<relaysw>3</relaysw>
</controller>
<relay_pos>
  <name>K03_0</name>
  <relay_position>0</relay_position>
</relay_pos>
<relay_pos>
  <name>K03_1</name>
  <relay_position>1</relay_position>
</relay_pos>
<relay_pos>
  <name>K03_2</name>
  <relay_position>2</relay_position>
</relay_pos>
<relay_pos>
  <name>K03_3</name>
  <relay_position>3</relay_position>
</relay_pos>
<relay_pos>
  <name>K03_4</name>
  <relay_position>4</relay_position>
</relay_pos>
<relay_pos>
  <name>K03_5</name>
  <relay_position>5</relay_position>
</relay_pos>
<relay_pos>
  <name>K03_6</name>
  <relay_position>6</relay_position>
</relay_pos>
</relay>
```

```

<relay>
  <name>K04</name>
  <type>AG_87204C_26_5GHZ_4W</type>
  <controller>

<functional_name>F_Relay_Controller</functional_name>
  <dut_pos>4</dut_pos>
  <module>4</module>
  <bank>4</bank>
  <relaysw>4</relaysw>
</controller>
<relay_pos>
  <name>K04_0</name>
  <relay_position>0</relay_position>
</relay_pos>
<relay_pos>
  <name>K04_1</name>
  <relay_position>1</relay_position>
</relay_pos>
<relay_pos>
  <name>K04_2</name>
  <relay_position>2</relay_position>
</relay_pos>
<relay_pos>
  <name>K04_3</name>
  <relay_position>3</relay_position>
</relay_pos>
<relay_pos>
  <name>K04_4</name>
  <relay_position>4</relay_position>
</relay_pos>
<relay_pos>
  <name>K04_5</name>
  <relay_position>5</relay_position>

```

```

        </relay_pos>
        <relay_pos>
            <name>K04_6</name>
            <relay_position>6</relay_position>
        </relay_pos>
    </relay>
    <relay>
        <name>K05</name>
        <type>AG_87104B_20_GHZ_4W</type>
        <controller>

<functional_name>F_Relay_Controller</functional_name>
        <dut_pos>5</dut_pos>
        <module>5</module>
        <bank>5</bank>
        <relaysw>5</relaysw>
    </controller>
    <relay_pos>
        <name>K05_0</name>
        <relay_position>0</relay_position>
    </relay_pos>
    <relay_pos>
        <name>K05_1</name>
        <relay_position>1</relay_position>
    </relay_pos>
    <relay_pos>
        <name>K05_2</name>
        <relay_position>2</relay_position>
    </relay_pos>
    <relay_pos>
        <name>K05_3</name>
        <relay_position>3</relay_position>
    </relay_pos>
    <relay_pos>

```



```

        <name>K05_4</name>
        <relay_position>4</relay_position>
    </relay_pos>
    <relay_pos>
        <name>K05_5</name>
        <relay_position>5</relay_position>
    </relay_pos>
    <relay_pos>
        <name>K05_6</name>
        <relay_position>6</relay_position>
    </relay_pos>
</relay>
<relay>
    <name>K06</name>
    <type>HP_8765D_40_GHZ_2W</type>
    <controller>

<functional_name>F_Relay_Controller</functional_name>
    <dut_pos>6</dut_pos>
    <module>6</module>
    <bank>6</bank>
    <relaysw>6</relaysw>
</controller>
<relay_pos>
    <name>K06_0</name>
    <relay_position>0</relay_position>
</relay_pos>
<relay_pos>
    <name>K06_1</name>
    <relay_position>1</relay_position>
</relay_pos>
<relay_pos>
    <name>K06_2</name>
    <relay_position>2</relay_position>

```

```

</relay_pos>
<relay_pos>
  <name>K06_3</name>
  <relay_position>3</relay_position>
</relay_pos>
<relay_pos>
  <name>K06_4</name>
  <relay_position>4</relay_position>
</relay_pos>
<relay_pos>
  <name>K06_5</name>
  <relay_position>5</relay_position>
</relay_pos>
<relay_pos>
  <name>K06_6</name>
  <relay_position>6</relay_position>
</relay_pos>
</relay>
<relay>
  <name>K07</name>
  <type>AG_87106D_40_GHZ_6W</type>
  <controller>

<functional_name>F_Relay_Controller</functional_name>
  <dut_pos>7</dut_pos>
  <module>7</module>
  <bank>7</bank>
  <relaysw>7</relaysw>
</controller>
<relay_pos>
  <name>K07_0</name>
  <relay_position>0</relay_position>
</relay_pos>
<relay_pos>

```

```

        <name>K07_1</name>
        <relay_position>1</relay_position>
    </relay_pos>
    <relay_pos>
        <name>K07_2</name>
        <relay_position>2</relay_position>
    </relay_pos>
    <relay_pos>
        <name>K07_3</name>
        <relay_position>3</relay_position>
    </relay_pos>
    <relay_pos>
        <name>K07_4</name>
        <relay_position>4</relay_position>
    </relay_pos>
    <relay_pos>
        <name>K07_5</name>
        <relay_position>5</relay_position>
    </relay_pos>
    <relay_pos>
        <name>K07_6</name>
        <relay_position>6</relay_position>
    </relay_pos>
</relay>
<relay>
    <name>K08</name>
    <type>AG_87104B_20_GHZ_4W</type>
    <controller>

<functional_name>F_Relay_Controller</functional_name>
    <dut_pos>8</dut_pos>
    <module>8</module>
    <bank>8</bank>
    <relaysw>8</relaysw>

```

```
</controller>
<relay_pos>
  <name>K08_0</name>
  <relay_position>0</relay_position>
</relay_pos>
<relay_pos>
  <name>K08_1</name>
  <relay_position>1</relay_position>
</relay_pos>
<relay_pos>
  <name>K08_2</name>
  <relay_position>2</relay_position>
</relay_pos>
<relay_pos>
  <name>K08_3</name>
  <relay_position>3</relay_position>
</relay_pos>
<relay_pos>
  <name>K08_4</name>
  <relay_position>4</relay_position>
</relay_pos>
<relay_pos>
  <name>K08_5</name>
  <relay_position>5</relay_position>
</relay_pos>
<relay_pos>
  <name>K08_6</name>
  <relay_position>6</relay_position>
</relay_pos>
</relay>
<relay>
  <name>K09</name>
  <type>AG_87204C_26_5GHZ_4W</type>
<controller>
```

```

<functional_name>F_Relay_Controller</functional_name>
    <dut_pos>9</dut_pos>
    <module>9</module>
    <bank>9</bank>
    <relaysw>9</relaysw>
</controller>
<relay_pos>
    <name>K09_0</name>
    <relay_position>0</relay_position>
</relay_pos>
<relay_pos>
    <name>K09_1</name>
    <relay_position>1</relay_position>
</relay_pos>
<relay_pos>
    <name>K09_2</name>
    <relay_position>2</relay_position>
</relay_pos>
<relay_pos>
    <name>K09_3</name>
    <relay_position>3</relay_position>
</relay_pos>
<relay_pos>
    <name>K09_4</name>
    <relay_position>4</relay_position>
</relay_pos>
<relay_pos>
    <name>K09_5</name>
    <relay_position>5</relay_position>
</relay_pos>
<relay_pos>
    <name>K09_6</name>
    <relay_position>6</relay_position>

```

```

        </relay_pos>
    </relay>
    <relay>
        <name>K10</name>
        <type>HP_8765D_40_GHZ_2W</type>
        <controller>

<functional_name>F_Relay_Controller</functional_name>
        <dut_pos>10</dut_pos>
        <module>10</module>
        <bank>10</bank>
        <relaysw>10</relaysw>
    </controller>
    <relay_pos>
        <name>K10_1</name>
        <relay_position>1</relay_position>
    </relay_pos>
    <relay_pos>
        <name>K10_2</name>
        <relay_position>2</relay_position>
    </relay_pos>
</relay>
<routing>
    <functional_name>CH 1</functional_name>
    <config>
        <name>Dut Input</name>
        <relay_pos>K01_1, K02_1</relay_pos>
    </config>
</routing>
<routing>
    <functional_name>CH 2</functional_name>
    <config>
        <name>Dut Input</name>
        <relay_pos>K01_2, K03_1</relay_pos>

```

```
</config>
</routing>
<routing>
  <functional_name>CH 3</functional_name>
  <config>
    <name>Dut Input</name>
    <relay_pos>K01_2, K03_2</relay_pos>
  </config>
</routing>
<routing>
  <functional_name>CH 4</functional_name>
  <config>
    <name>Dut Input</name>
    <relay_pos>K01_2, K04_3</relay_pos>
  </config>
</routing>
<routing>
  <functional_name>CH 5</functional_name>
  <config>
    <name>Dut Input</name>
    <relay_pos>K01_2, K05_2</relay_pos>
  </config>
</routing>
<routing>
  <functional_name>CH 6</functional_name>
  <config>
    <name>Dut Output</name>
    <relay_pos>K01_3, K06_2</relay_pos>
  </config>
</routing>
<routing>
  <functional_name>CH 7</functional_name>
  <config>
    <name>Dut Output</name>
```

```
        <relay_pos>K01_3, K07_2</relay_pos>
    </config>
</routing>
<routing>
    <functional_name>CH 8</functional_name>
    <config>
        <name>Dut Output</name>
        <relay_pos>K01_3, K08_2</relay_pos>
    </config>
</routing>
<routing>
    <functional_name>CH 9</functional_name>
    <config>
        <name>Dut Output</name>
        <relay_pos>K01_3, K09_2</relay_pos>
    </config>
</routing>
<routing>
    <functional_name>CH 10</functional_name>
    <config>
        <name>Dut Output</name>
        <relay_pos>K01_3, K10_2</relay_pos>
    </config>
</routing>
</Relay_matrix>
</jigs>
</test_setup>
```