KONGSBERG
Precision Cutting Systems

Bachelor's thesis

Kongsberg Table Tracker

University of South-Eastern Norway

Faculty of Technology, Natural Sciences and Maritime Sciences
Campus Kongsberg

This page was intentionally left blank.

Course: TS3000 Bacheloroppgave
Date:
Title: Kongsberg Table Tracker

Project group: 13
Group members:
Ole William Skistad Huslende
Tormod Smidesang
Elvin Andreas Pedersen
Magnus Trillhus Olsnes

Internal supervisor:
Olaf Hallan Graven

External supervisors:
Martin Kvalbein
Håkon Weirud
Anders Hove

Project partner:
Kongsberg Precision Cutting Systems

Acknowledgments

We would like to sincerely thank our external supervisors, Lars-Ole Aamodt, Anders Hove, Martin Kvalbein, and Håkon Weirud for providing us with the resources and opportunity to complete this project. We have been extremely lucky to have multiple supervisors that has shown real interest in this bachelor project. We are especially grateful for their continuous support and guidance throughout the process. Also a quick thanks to Heidi Sundby for providing sustenance at Kongsberg Precision Cutting Systems.

We also give our heartfelt thanks to our internal supervisor, Olaf Hallan Graven, for sharing his guidance throughout the entirety of the Kongsberg Table Tracker project. His insight, experience, and thoughtful advice during our many meetings have been of great importance to our progress and learning, especially within the academic field.

Finally, we thank the University of South-Eastern Norway for providing us with the opportunity to undertake this research. We also acknowledge the support of the faculty, who have provided us with the resources and knowledge necessary to succeed.

Abstract

The topic of this report is the research and development of a prototype application that provides remote monitoring of cutting tables. This is to improve the workflow of production facilities that utilize these types of machines. This was done by using an existing Application Programming Interface (API) that collects information from a table. The information was aggregated from multiple sources into a functioning application that displays relevant information to the user. The result was a working prototype with an intuitive user interface as well as an architecture that supports future expansion. This application has great future potential for both managers and operators, while also contributing to ongoing research and development.

This page was intentionally left blank.

Contents

List of Figures

List of Tables

List of Code Segments

Glossary

.NET Software framework by Microsoft. 22–26, 35, 38, 45, 56, 58, 73, 82, 84

API Application Programming Interface. 4, 18, 26, 39, 43, 46, 59, 81

C# C Sharp - general-purpose high-level programming language. 21, 22, 35, 81

CLI Command Line Interface. 21, 22

CNC Computer Numerical Control. 19

CSS Cascading Style Sheet. 32

DCS Digital Cutting System. 16

DNS Domain Name System. 25, 76

GUI Graphical User Interface. 20–23, 30, 41, 46

HTML HyperText Markup Language. 23, 32

HTTP HyperText Transfer Protocol. 26, 56

IDE Integrated Development Environment. 22, 32, 35, 36

IP Internet Protocol. 75, 76

iPC i-cut Production Console. 16–20, 36, 54, 55, 69, 81, 82

JSON JavaScript Object Notation. 21, 24–26, 46, 56, 58, 59, 75

Kongsberg HUB Service that runs on all computers where i-cut Production Console is installed, provides an API for interacting with cutting tables. 16, 20, 21, 39, 41, 43, 46, 51, 52, 54–58, 65, 74, 75, 81, 82, 84

KPC Kongsberg Precision Cutting. 17

KPCS Kongsberg Precision Cutting Systems. 14, 16, 18–20, 31–33, 35, 36, 38, 40, 45, 54–58, 61, 78, 79, 81, 84

KPU Kongsberg Production Unit. 16, 18

KTT Kongsberg Table Tracker. 19, 20, 33, 46, 47, 49, 54, 61, 64, 65, 84, 109

LAN Local Area Network. 83

MAUI Multi-platform App User Interface. 21–24, 35, 38, 45, 56, 73, 82

MultiCam A U.S.-based company that produces CNC cutting machines. Acquired by Kongsberg Precision Cutting Systems in 2021 to expand their cutting solutions portfolio. 19

MVVM Model-View-ViewModel. 21, 24, 43, 45, 52

PDF Portable Document Format. 35

POCO Short for Plain Old Class Object, a class that does not have any methods, just pure data. 25

REST Representational State Transfer. 18, 26, 46

RPN Risk Priority Number. 31, 80

SSE Server-Sent Events. 26, 43, 58

UI User Interface. 17, 21, 22, 24, 45, 46, 51, 52, 84

URL Uniform Resource Locator. 26

USN University of South-Eastern Norway. 32

UX User Experience. 21–23, 30, 46, 47, 49, 65

XAML Extensible Application Markup Language. 21, 22, 24, 46, 61, 62, 64, 76, 107

1 Introduction

1.1 Overview **OH |*TS***

In recent years, the packaging industry has undergone a technological revolution, driven by the surge in e-commerce and increasing consumer demand for delivered goods. The need for innovative, efficient, and sustainable packaging solutions has never been greater. One company that aims to meet this demand is Kongsberg Precision Cutting Systems (KPCS). They have been making creative and efficient cutting tables used in the packaging, label, and manufacturing industry for decades. Their standalone production devices allow users to upload design files and precisely cut a wide range of materials, enabling greater flexibility and creativity in the packaging process. However, a current limitation is that these standalone cutting tables are not remotely accessible or integrated with each other. The lack of monitoring and management of the production process creates challenges. Therefore, this report aims to explore how the problem can be solved by enabling remote tracking and interaction with the production units. This will enhance oversight, coordination and overall efficiency for the production.

1.2 Group Members **MO |*EP***

A project group was established shortly after the semester began. Two students initiated the group formation and recruited two additional members. The group consisted of four students by the time the project started. One member participated remotely the first month due to an exchange program abroad, which required early planning for digital collaboration.
KPCS was selected as the external company partner. The connection was made through one of the group members who works there, and the company provided a suitable assignment for the bachelor project.
To ensure effective collaboration, group roles and responsibilities were defined early. This allowed the team to begin work efficiently and manage tasks in a structured manner.

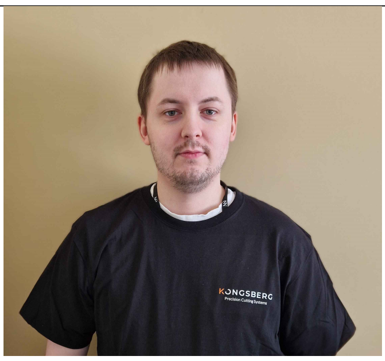
	Name: Elvin Andreas Pedersen **Initials:** EP **Discipline:** Software Engineer **Project Role:** GUI developer
	Name: Ole William Skistad Huslende **Initials:** OH **Discipline:** Software Engineer **Project Role:** Scrum-master and fullstack developer
	**Name:** Tormod Smidesang **Initials:** TS **Discipline:** Software Engineer **Project Role:** Git responsible and fullstack developer
	Name: Magnus Trillhus Olsnes **Initials:** MO **Discipline:** Software Engineer **Project Role:** Documentation lead and test developer

Table 1: Overview of Group Members

1.3 Initials In Section Headers TS |*OH*

Inspired by the reports of bachelor projects from previous years, we have added the initials of the person who wrote a particular (sub)section to the header of said (sub)section. Another set of initials in *italics* to the right of the aforementioned initials indicates who has proofread the same (sub)section. A subsection that lacks these initials should be assumed to be the responsibility of the set of people listed in its parent section header.

2 Domain EP |*MO*

Technical terms essential for understanding this report are further explained in the "Glossary" section at the beginning of the report. Readers can refer to this for clarification if needed. The domain of this report is Digital Cutting System (DCS) and will mostly refer to KPCS and their solutions. KPCS has defined what they call a Kongsberg Production Unit (KPU), which consists of four main components. First is the Kongsberg cutting table. Second is a Windows PC, which contains the last two components, namely Kongsberg HUB and i-cut Production Console (iPC). The following chapters will explain the components in more detail, as they are essential for a complete understanding of the context later on.

2.1 Cutting Table

A cutting table operates in a wide range of industrial sectors such as corrugated packaging production, protective packaging production, display production, signage production, labels & decal production, and sample making. [4] A cutting table refers to a flatbed machine controlled by a computer. It is designed to cut, crease, or engrave on different types of materials. [5] [6] Figure 1 shows a picture of the Kongsberg Ultimate table, the most recent addition to the KPCS collection. A table usually consists of a flat and durable surface in which the material is placed. This surface can move or rotate on some models and is especially useful for factory automation. Above the cutting surface is the tool head, which works according to the instructions provided by iPC. It is mounted to a beam that enables movement along the x-axis of the table. The tool head moves along the y-axis of the table by traversing the beam. The table has a durable surface that is resistant to cutting, but can also have something called a cutting mat or sacrificial underlay. This is an extra protective layer that prevents excessive wear and tear after continuous use. It can cover the entire, or just parts of the cutting surface.

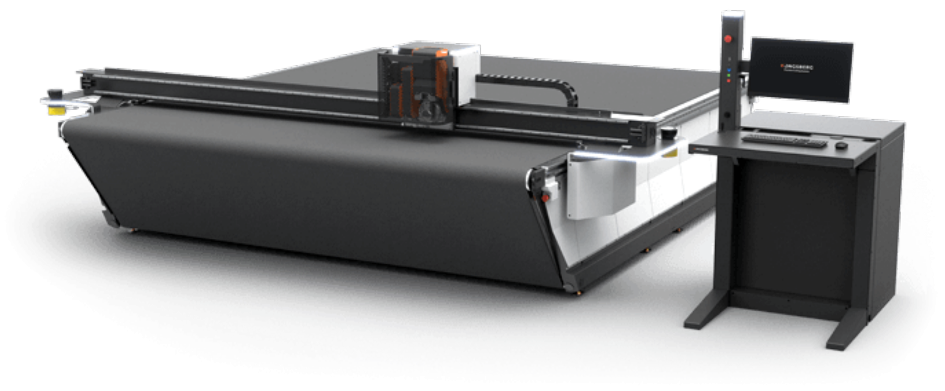


Figure 1: Kongsberg Ultimate 64 Source: [1]

Some of the most common materials used on cutting tables are corrugated materials, folding carton & paper, cardboard, rigid board, foam materials, aluminum & ACM and wood-based materials. [4] In order to handle such a versatile set of materials, it is also necessary to have the right tools for the job. Modern cutting tables often support a wide range of different tools. Knives are used to directly cut materials, and some cut using vibration. Creasing tools use downforce to compress the material, creating imprints or folds. Milling tools cut by rotating at high speeds and chip away at the material. [7] Drilling tools are used to create holes, often for screws or joint connectors. The main difference between the two are that drilling moves along the z-axis and milling can move along all three axes. There are also a multitude of other specialized tools. [8] The cutting table can also utilize vacuum to move and secure the material while operating. This helps to ensure a higher level of precision and stability. iPC dynamically adjust the vacuum strength based on what type of material is used. [9] The Kongsberg Precision Cutting (KPC) tables also have a smart modular design, which provides more flexibility. It can have multiple tools mounted at the same time and swap them out easily when needed. Some tables also support additions like roll-to-roll or pallet-to-pallet automation by using a feeder/stacker configuration, greatly increasing production efficiency. [10]

2.2 i-cut Production Console

With the Kongsberg tables follows an application/User Interface (UI) called iPC, see Figure 2. It works as the control software for the table and UI for the operator. iPC allows the operator to upload job files or even create simple designs and layouts directly within the software. The operator can customize runs (one instance of running a job) and calibrate the table by setting up materials and settings for each job. Each job can be run on it's own, or combined into batches or layouts. In short, the whole production can be controlled and run from iPC. Every iPC also comes with a Kongsberg HUB. [11] [2]

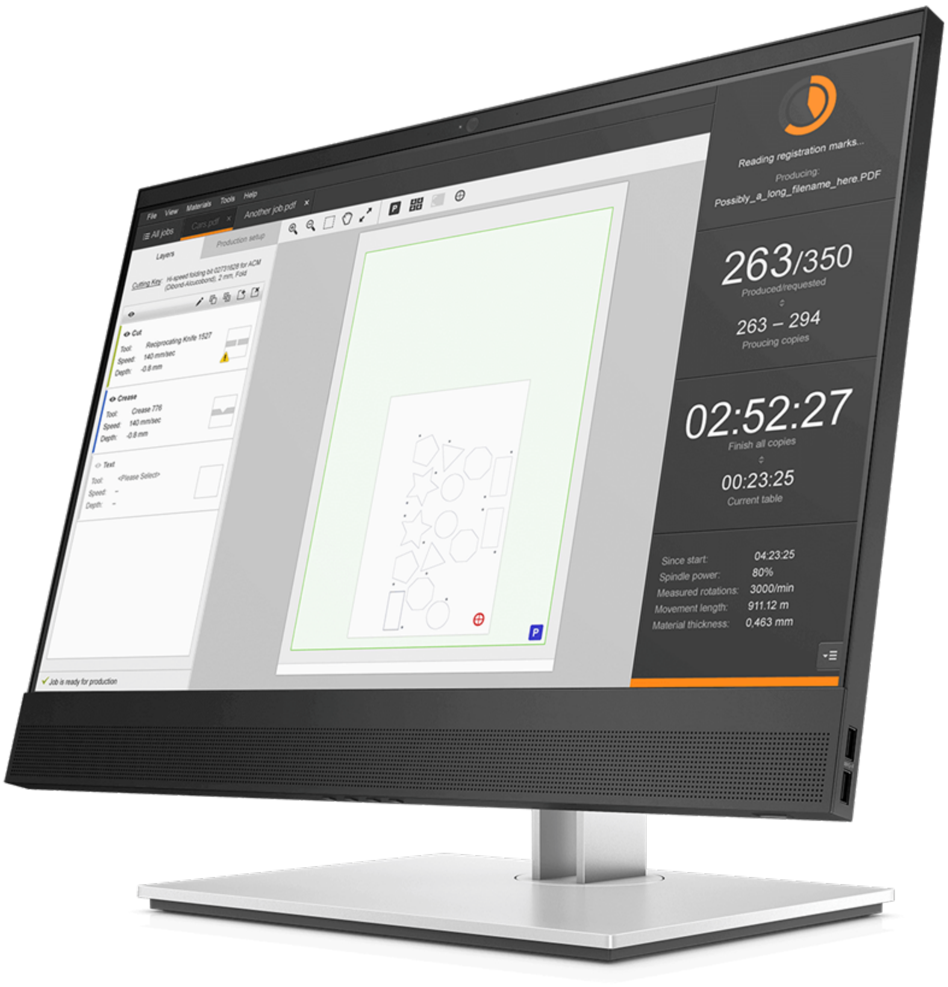


Figure 2: i-cut Production Console, Source: [2]

2.3 Kongsberg HUB

Disclaimer: The source of the information in this section is not disclosed in this report, in accordance with the wishes of KPCS.

Cutting tables with iPC 3.0 and newer all have a Kongsberg HUB - Representational State Transfer (REST) API running in the background. The HUB stores all of the job information from the table. Through the HUB it becomes possible to fetch information from the table externally and display it outside of the KPU. Information of interest includes table properties, resources, job lists, and statistics. It also provides options for interaction with the table. Some external interactions include uploading jobs, deleting jobs, and setting certain configurations. [11]

3 Problem **EP |*MO***

This chapter will explain more of the background regarding the Kongsberg Table Tracker (KTT) project. The first section will be a more in-depth introduction to KPCS, which is defined as the customer in this report. The second section deals with a problem KPCS is facing, and the third contains the definition of the problem this report aims to solve.

3.1 Kongsberg Precision Cutting Systems **MO |*EP***

KPCS became an independent business in 2021, though its origins date back to 1965 in Kongsberg, Norway. Before becoming a company, the technology that later inspired KPCS was developed as part of the former weapon factory in Kongsberg. At that time, machines used punch cards for metal cutting, and one incorrect punch could cause expensive mistakes. To avoid this, engineers created a drawing table system to plan the cuts before entering data into the machine. This was important because the metal plates used for cutting could cost up to 10,000 NOK in the 1960s a large amount of money at that time. This solution reduced errors and costs and became the foundation for a new business idea.
In 1965, some individuals saw the potential and started a company focused on these drawing tables. From there, the technology evolved into the digital cutting solutions used today. [12]

The company combines expertise, technology, and innovation to deliver high-quality digital cutting and Computer Numerical Control (CNC) solutions. As the first diversified provider of both digital and CNC cutting machines, KPCS manufactures and distributes two of the industry's leading brands: Kongsberg and MultiCam. [12]

The Kongsberg product line offers highly durable and precise digital cutting solutions for the packaging, signage, display, and manufacturing industries. These solutions help businesses optimize production by increasing speed, safety, and efficiency without restricting creative potential. While research and development remain in Kongsberg, Norway, manufacturing takes place in Brno, Czech Republic. The company also maintains a strong global service network with experienced engineers and certified service partners to ensure continuous production. [12]

MultiCam, known for its U.S.-made cutting equipment, provides a range of CNC routers, digital cutters, and waterjet cutting machines. These machines serve various industries, including signage, digital finishing, aerospace, automotive, sheet metal, woodworking, and plastics fabrication. [12]

KPCS operates under the ownership of OpenGate Capital and is headquartered in Ghent, Belgium. The company also has a North American head office in Ohio, USA. [13] [4]

3.2 Existing Software and Limitations **EP |*MO***

The current software used by Kongsberg cutting tables is called iPC, and is defined in Chapter 2.2. The iPC provides an intuitive overview and easy control over the Kongsberg table; however, it has one noticeable drawback. What it currently lacks is a way for the user to efficiently receive information from the table remotely. In order to get the status or progress of the table, the user has to physically visit the table and check the production console. KPCS have recived request from their customers to provide a solution that allows them to access this information from a remote location.

It is important to mention that there are currently other solutions that can retrieve and display information from the table, but KPCS wishes to develop their own solution and not rely on third party tools.

3.3 Problem Statement EP |*MO*

Problem Definition

Currently, KPCS does not have a local solution that provides remote view of the iPC. For customers, this can affect the monitoring and management of the cutting tables. This project aims to build an application that remotely collects information from the iPC through the Kongsberg HUB KPCS will take over the application at the end of the semester. They expect it to meet a certain quality standard so that they can continue to develop and eventually release it to their customers.

Problem Impact

Without a remote monitoring solution, operators have to rely on less efficient methods to obtain information from the cutting tables. This can lead to wasted time and reduced productivity. By solving this problem, operators will gain more flexibility and control over their production. They can check on job progress/status from their office, rather than having to inspect the table in person. This will save them time, increase productivity, and is especially useful with multiple cutting tables involved in production. The KTT application should be considered a “quality of life” improvement for operators.

Project Scope

The scope of the project is to create a prototype application that collects and displays information from multiple Kongsberg tables via a Graphical User Interface (GUI). The application shall include a page with an overview of the tables and a page for a detiled view from each table where jobs and status is displayed. Further requirements will be defined in Chapter 7.

4 Theory

This chapter presents important theory and key concepts on which the project is based upon. Understanding them is essential to understand how the system is designed, implemented, and tested. The first section explains the framework for the application which serves as the structural backbone of the system. Following that is a short description of each programming language used within the Framework. The next section delves into GUI & User Experience (UX), and best practices. To provide a clear architectural understanding, there is also a section that explains what the MVVM (Model-View-ViewModel) design pattern is and why it is so important. After that comes a separate section regarding JavaScript Object Notation (JSON), which plays a central role in how data is structured, followed by networking principles relevant to how data is sent and retrieved from external systems, such as the Kongsberg HUB. The last section is about testing, highlighting strategies and tools used to ensure the reliability and maintainability of an application.

4.1 Framework EP |*MO*

.NET Multi-platform App User Interface (MAUI) is a framework that has evolved from Xamarin.Forms and focuses on cross-platform development. It is possible to build native apps on Windows, macOS, iOS, and Android in one single project, saving developers time and effort. [14] The .NET MAUI framework uses Extensible Application Markup Language (XAML) & C# as programming languages. When combined with the community tool kit and Model-View-ViewModel (MVVM) architecture (more about that later), building intuative and modern applications becomes possible, even for beginners. [15] .NET MAUI also supports the "Hot Reload" in Visual Studio, which means you can build and view a GUI in real time. [16]

4.2 Programming Languages EP |*MO*

XAML

XAML stands for eXtensible Application Markup Language and is created by Microsoft. It is used to build interfaces in applications [17] [18] and helps to separate UI and code logic. See Chapter 4.3 for more details on XAML.

C#

C#, pronounced C Sharp, is a Microsoft-developed programming language. It is used to build many types of programs, like desktop applications, mobile apps, websites, and games. C# is part of the .NET platform and acts as the main coding language. C# is an object-oriented language, which means it is based on objects and classes. This makes the code easy to organize and reuse, especially for bigger and more complex systems. C# has a simple and modern syntax, and is similar to other languages such as Java and C++.

4.3 Graphical User Interface EP |*MO*

Introduction

A User Interface is the connection for communication between a user and a digital system. There are different types of UIs, such as Command Line Interface (CLI), GUI, Touch UI,

Voice UI, Menu-driven UI, and many more. [19] A CLI can be challenging to learn, and to make computers more accessible to less technical users, GUI was developed.
A GUI is a visual layer that utilizes graphical elements such as pages, buttons, sliders, icons, etc., to present information to the user in a more intuitive way. Instead of manually entering commands into a CLI, the user interacts with visual components that represent different functions of the system. All commands are set and executed behind the scenes, and only the necessary elements are presented to the user. This makes the system appear less complex and can increase productivity because one click can trigger multiple commands and the learning curve for the system is more beginner friendly.
UX is an important key word in relation to GUI development. The user experience can be defined by looking at the system from a users perspective. Does the system provide a good or bad experience? Does the layout and navigation make sense? Are the colors creating confusion or do they enhance the system message? [19] [20]

Why does GUI matter?

One of the main reasons why a good GUI matters for an application is that it is one of the main deciding factors for the customer. In this day and age, there is no shortage of applications to choose from and the competition is fierce. In order to succeed in the targeted market, it is important to present a better user experience than the competition. Development speed and versitility can make or break a startup, and this is where .NET MAUI has several advantages. As mentioned, it gives the ability to target multiple platforms at once, all in one single project.

Developer tools and Enviorment

For .NET MAUI development, the usual choice for Integrated Development Environment (IDE) is ”Visual Studio” [21] as it is under the Microsoft umbrella, just like .NET MAUI. It is worth mentioning that there are other options out there, for example JetBrains ”Rider” [22]. The ”hot reload” within Visual Studio is a very useful tool as it lets you see changes in real time while developing. [16] When building a GUI, it is actually possible to do it in both C# and XAML, but the norm is to use xaml for pages and C# for code.

Project structure

Before diving deeper into the building block of a GUI in .NET MAUI, it is important to understand the basics of the project structure. When a new MAUI project is first generated, it will create three important XAML files along with their code-behind. The three files are called App.xaml, AppShell.xaml and MainPage.xaml. The App class (xaml file + code-behind) derives from the Application class and is the entry point of the application. What that means is that it sets the global resources and the root page. The code-behind of App handles the startup logic. AppShell is responsible for the navigation structure within the application and the code-behind is used for registering routes. The MainPage is the first content page that is loaded in the application. This is usually what is set as the root page mentioned earlier. [23]

Core Components

Navigation is the mechanism that allows the user to move between different pages. Well structures navigation is at the core of a good UX. Navigation can be done in different ways, such as stack-based, tab-based and shell-based. For stack-based, pages are pushed on to or popped from the stack. Tab-based includes a tab, typically a bar of sorts, with tabs as chilren. The more modern way is Shell navigation. It allows the developer to set routes for different pages, which can provide clean and easy navigation flow across the application. [24]

Pages are the screen that the user intaracts with and the most common is ContentPage (MainPage is of type ContentPage). The ContentPage has a property called "content", which is only able to hold one control. This will in most cases be a type of layout. If a project has the CommunityToolKit included, one can also use the toolkit:popup, perfect for smaller interactions with the user. [25] [26]

Layouts or containers are used to decide how elements (also known as UI controls) are placed inside a page. The usual layouts are StackLayout (vertical/horizontal), Absolute-Layout, Grid, and FlexLayout. See 3. The stacklayout takes each control and stacks them after one another. It is very useful as a container for smaller parts of the GUI. Up next is the AbsoluteLayout. As the name suggests, it uses absolute values to decide where the controls are placed. It can be useful for improving the cetain controls or parts in the GUI. A Grid is built on rows and columns much like a table. Controls can be places in the "coordinates" of the grid and also span over several rows or columns. The height and width can be static (numeric value), automatically fit the content (Auto), or take the remaining space on the page (*). Grid provides excellent control over the GUI, and is often used as a parent to other layouts. Lastly, there is the FlexLayout. It is very similar to the flexbox layout in HyperText Markup Language (HTML), and works like a stacklayout, but with the extra functionality of wrapping its elements in new rows or columns. This can be used on pages that display a list that vary in size. [3]

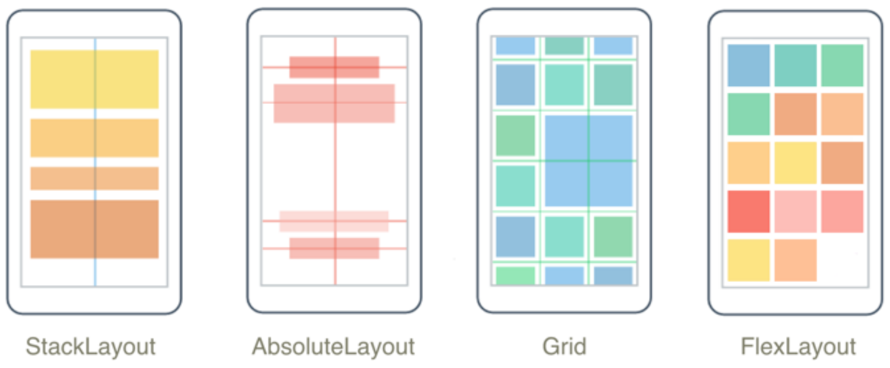


Figure 3: Illustration of layouts, Source: [3]

Controls is a term that can mean different things depending on the context, but generally speaking, it refers to smaller visual elements in a layout. The following is a list of some of the most widely used controls in .NET MAUI. [27]

- BoxView
- Button
- CheckBox
- CollectionView
- ContentView
- DatePicker
- Entry
- Frame
- Image
- ImageButton
- Label
- Picker
- ProgressBar
- RadioButton
- ScrollView
- Slider

4.4 Model-View-ViewModel OH |*TS*

The **Model-View-ViewModel** architectural pattern is widely used in software development to promote a clear separation between an application's UI and its underlying business logic. This separation enhances maintainability, testability, and scalability [28].

Components of MVVM

The **View** is the UI layer that presents data to the user and captures user interactions. In technologies like .NET MAUI, the view is often defined using XAML, allowing developers and designers to work on the UI separately from the underlying code.
The **Model** represents the application's data and business logic. It defines the data structures and implements the operations to manage and manipulate this data. The model is unaware of the UI.
The **ViewModel** serves as an intermediary between the Model and the View. It exposes data from the Model in a form that the View can easily consume. It handles user commands, processing input and updating the Model accordingly. The ViewModel facilitates data binding, ensuring synchronization between the Model and the View.

Benefits of MVVM

One key benefit of MVVM is the separation of concerns. By decoupling the UI from the business logic, MVVM allows developers to modify the UI without impacting the core logic and vice versa. This leads to cleaner and more organized codebases.
Another important benefit is enhanced testability. With the business logic encapsulated in the ViewModel, developers can write unit tests for this layer without involving the UI, which leads to more reliable and maintainable code.
MVVM also allows for improved collaboration. Developers and designers can work concurrently on the same application. Designers can focus on creating the UI in XAML, while developers implement the logic in the ViewModel and Model. This streamlines the development process and helps avoid conflicts.

4.5 JSON TS |*OH*

JSON, short for JavaScript Object Notation, is a data format for easy storage and transmission of data. It can be easily read and modified by both people and software. In JSON, data is stored in name/value pairs, separated by commas and organized into objects with curly brackets. [29]

.NET has built in support for converting data to and from JSON format (processes called serialization and deserialization, respectively) [30]. While the supported data types are normally limited to primitives such as number types, strings, booleans and POCOs, .NET makes it possible to manually add support for other types through JsonSerializerOptions [31]

4.6 Networking **TS |*OH***

IP addresses

An IPv4 address is a unique identifier for a device on a network. It consists of 32 bits which are usually represented in dotted decimal format (example: 192.168.100.43). An IP address is divided into a host part that identifies the device and a network part that identifies the network. These parts are distinguished by using a subnet mask. Subnet masks also consist of 32 bits, but unlike IP addresses, they have all their bits set to 1 until a certain point after which all the bits are set to 0. The left part of the subnet mask (where the bits are all 1) is the network part of the IP address. The right part (where the bits are all 0) is the host part of the IP address. An example subnet mask 255.255.255.0 has the first 24 bits set to 1 and the remaining 8 bits set to 0 (a network with this subnet mask is sometimes referred to as a class C network, which can have at most 254 devices). This subnet mask can also be represented with a forward slash followed by the amount of 1-bits. Combining the examples used in this section, the IP address and subnet mask could collectively be represented this way: 192.168.100.43/24. [32]

DNS

A Domain Name System (DNS) server keeps a record of hostnames and their associated IP address. It handles DNS requests where someone wants the IP address associated with a hostname and returns the hostname if it knows it [33]. A reverse DNS lookup is the same thing, but the other way around [34].

HTTP

The HyperText Transfer Protocol (HTTP) is an application-layer protocol used for communication between clients and servers over the internet. It operates on a request-response model, where a client, such as a web browser or some other application, sends an HTTP request to a server, which then responds with the requested resources or data. HTTP is stateless, meaning each request is independent, and it supports various methods like GET, POST, PUT, and DELETE to perform different actions. [35]

In .NET, HTTP communication is primarily handled through the HttpClient class, part of the System.Net.Http namespace, which provides an API for sending HTTP requests and processing responses. HttpClient supports asynchronous operations, making it well-suited for applications where responsiveness is critical. HttpClient can be configured with custom headers, timeouts, and authentication credentials to interact with REST APIs or other web services. This enables .NET applications to perform secure and efficient HTTP-based communication. [36]

Server-Sent Events (SSE)

Server-Sent Events (SSE) is a technology to receive updates automatically from a server through an HyperText Transfer Protocol (HTTP) connection [37]. In .NET, it is possible to subscribe to Server-Sent Events (SSE) using the HttpClient.GetStreamAsync method [38]

REST API

REST (Representational State Transfer) is a standardized way for applications to communicate over the internet. It uses HTTP methods (GET, POST, etc.) to perform operations on resources, or end-points, which are identified by a Uniform Resource Locator (URL). A REST API enables applications to send requests to a server (such as retrieving or updating data) and receiving responses, typically in JSON format. [39]

4.7 Testing Code MO |*EP*

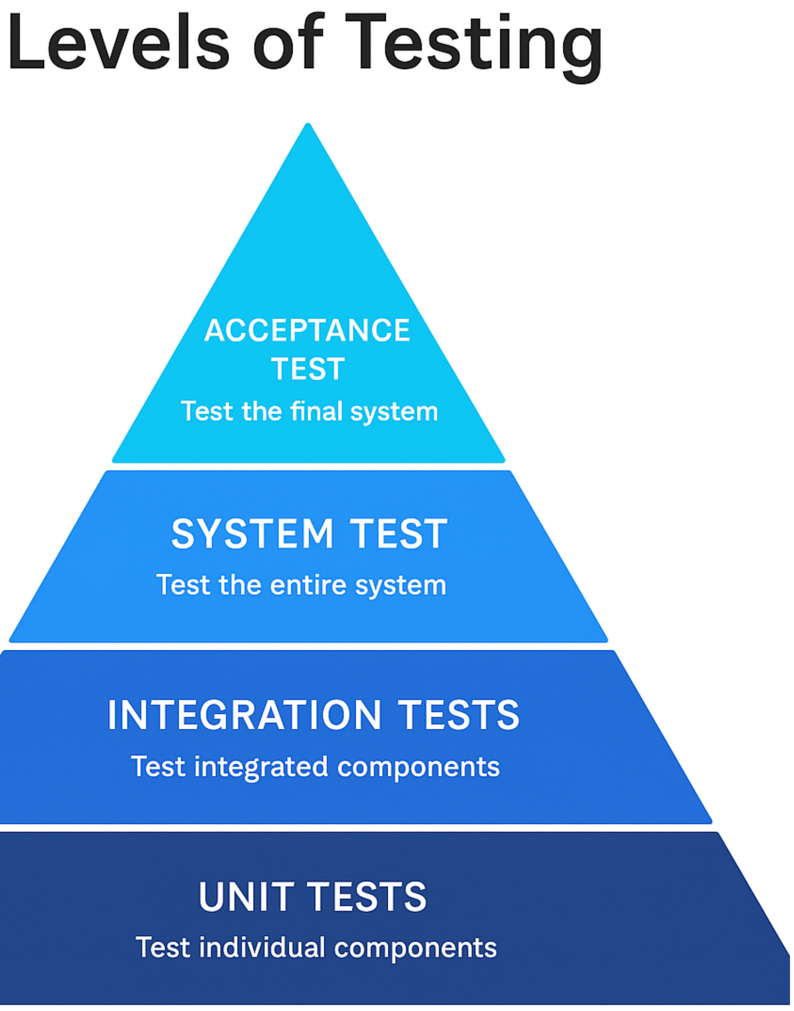


Figure 4: Levels of testing

There are many ways to test code, but the four main testing strategies are unit testing, integration testing, system testing, and acceptance testing. These are the most common and cover different levels of the software. As seen in the image above (4), these four levels are often shown as a pyramid, where unit tests are at the bottom and acceptance tests are at the top. There are also other types of testing, such as performance testing, security testing, and usability testing. However, the four shown in the image are the most important and commonly used in most projects.

Unit testing MO |*EP*

Unit testing is a software testing method where individual components or functions of a program are tested separately to ensure they behave as expected. It is one of the first levels of testing and is typically written and maintained by developers during development. Unit tests focus on small, isolated pieces of code, such as functions or methods, using known inputs to verify expected outputs.

Unit testing helps catch bugs early, improves code reliability, and makes future changes to the codebase safer. Automated testing frameworks like JUnit, NUnit, and PyTest are often used to streamline the process. Good unit tests are fast, repeatable, and independent from each other [40].

Integration testing MO |*EP*

Integration testing is a software testing level that focuses on verifying how different modules or components of a system work together. While unit testing checks the internal correctness of individual units, integration testing ensures that the interfaces and interactions between those units function as expected.
This type of testing is important for identifying issues such as incorrect data handling between modules, interface mismatches, or unexpected side effects. It is typically performed after unit testing and before system testing. Common approaches include top-down, bottom-up, and big bang integration.
By using integration testing, development teams can find problems that only appear when modules are combined, which helps improve the reliability of the full system [41].

System Testing MO |*EP*

System testing is a type of software testing where the complete system is tested as a whole. It is usually done after unit testing and integration testing. The goal is to check if the full software works as it should, based on the requirements [42].

This type of testing includes both functional and non-functional checks. Functional testing makes sure that features do what they are supposed to do, while non-functional testing may include performance, security, and usability.
System testing is important because it shows how all parts of the application work together. It also helps to find problems that only happen when everything is running at the same time.

System testing is also important for identifying gaps between what the system does and what the user expects. It gives a complete picture of the software quality and is often one of the final steps before acceptance testing [43].

Acceptance testing **MO |*EP***

Acceptance testing is a type of software testing done to check if the system meets the needs of the user or customer. It is usually the last step before the software is released. The goal is to make sure the software works well in real-world situations and does what it is supposed to do [44].

This kind of testing is often done by the customer, the end user, or a testing team. They test the whole system by using it like a normal user would. If the system passes the acceptance test, it means the software is ready to be used.

Regression testing **MO |*EP***

Regression testing is a type of software testing done to make sure that new changes in the code do not break parts of the system that were already working. It is often done after a bug fix, feature update, or other code change. The goal is to check that the rest of the system still behaves correctly.
These tests can be done manually or automatically. In many projects, important tests are repeated to confirm that everything still works as before. This helps avoid introducing new problems when making updates [45].

5 Project Management

5.1 Organizational Structure MO |*EP*

The project team has four members. Each member has their own main responsibilities, but all members contribute to the software development process. Since Scrum is used, the team is structured to stay organized and to make sure everyone knows their responsibilities.

Elvin is the GUI developer, which means he is responsible for the graphics and UX.

Ole William Is the full-stack developer and Scrum Master. He makes sure the Scrum principles are followed, sprints run properly, and progress is tracked. This role also helps facilitate discussions and keeps the workflow smooth.

Magnus oversees documentation, which means he makes sure reports, technical documents, and important notes are well written and up-to-date. He is also responsible for ensuring that the documentation is delivered on time.

Tormod focuses mainly on software development and GitHub, and also acts as Product Owner. This role includes setting specific goals for each Scrum sprint.

Group Dynamic MO |*EP*

The group has a positive and effective working relationship. Members cooperate well and support each other when needed. There are no major problems, and communication stays clear throughout the project. Important tasks are understood, and regular meetings help keep the work organized and on track. Each member understands their role and follows the plan.
The group is split into two parts. Tormod and Ole William mostly work on the backend, while Magnus and Elvin work on the GUI. The two groups talk often and have meetings. This helps the project come together.

Team Building MO |*EP*

All group members knew each other before the project started, since we had been in the same class for two and a half years. Because of this, we did not focus much on teambuilding in the beginning. Teambuilding is often important for new groups, as it helps members cooperate and understand each other better, which can improve the group's dynamic and effectiveness.
At first, we postponed teambuilding because we were unsure what kind of activity we should do. Later, one group member suggested that we could play a video game together. The group agreed, and we played *World of Warcraft* together a couple of times. This helped strengthen our teamwork in an informal way.

5.2 Supervisor Communication

Internal MO |*EP*

The group was assigned an internal supervisor from the school. He had experience in computer science and engineering and provided useful feedback and advice throughout the project. Communication with Olaf happened mainly through email, and his replies were always fast and helpful. Weekly meetings were held every Wednesday. During these meetings, Olaf supported the group with many parts of the project, including documentation, the application, presentations, and more.

External OH |*TS*

KPCS provided multiple external supervisors from their R&D division in Teknologiparken. They also provided each group member an office as well as conference rooms available for booking. Thursdays and Fridays are spent at the office and meetings with the external supervisors are held every Friday. The schedule is decided prior to the meeting and the content varies from code reviews, functionality requests or changes to make the application more user friendly. Supervisors help solve challenges that arise during the development process. Direct communication with experienced developers makes for a good source of valuable feedback.

5.3 Project Risk Analysis MO |*EP*

In terms of project management, things like missed deadlines, team conflicts, and changing requirements could slow us down. To prevent this, good communication, proper planning, and staying ahead of potential problems are key.
Risk such as software bugs, security vulnerabilities, and integration issues could affect quality. These can be managed through testing, code reviews, and quality assurance.

Risk calculation

The Risk Priority Number (RPN) method is used to help measure the level of risk in a project. [46] It gives a number that is calculated by multiplying three factors:

- **Probability** shows how likely it is that the risk will happen.
- **Impact** shows how big the problem will be if the risk happens.
- **Detection** shows how easy or hard it is to notice or stop the risk before it becomes a problem.

Detection is important because some risks are easy to prevent, while others are harder to catch. If a risk is hard to detect, it is more dangerous, even if it does not happen often. The RPN is calculated like this:

$$\textbf{RPN} = \textbf{Probability} \times \textbf{Impact} \times \textbf{Detection}$$

The table below shows how RPN values are used to decide the overall risk level.

RPN Range	Overall Risk Level
1–14	LOW
15–29	MODERATE
30–49	HIGH
50 and above	EXTREME

Table 2: RPN value ranges and corresponding risk levels

Risk	Potential Consequences	Probability (1–5)	Impact (1–5)	Detection (1–5)	RPN	Overall Risk (RPN)
KPCS goes bankrupt	Loss of client	1	5	2	10	LOW
Missed deadline	Project delay	2	4	2	16	MODERATE
Team conflict	Inefficient working enviroment	1	3	3	9	LOW
Requirements changes	Extra workload	2	3	3	18	MODERATE
Legal issues	Project shutdown	1	5	2	10	LOW
Software bugs during development	Unstable system during testing	3	3	2	18	MODERATE
Software bugs after release	Unstable system for end-users	2	4	3	24	MODERATE

Table 3: Project Risk table

5.4 Website EP |*MO*

The project website is built using HTML and Cascading Style Sheet (CSS) in Webstorm IDE. The HTML is built using semantic elements, where divs are replaced with proper elements such as header, nav, section and article. This makes the documents more organized and flexible for future modifications. The CSS is initially generatet using AI, and further modified to suit out standards. Classes are used in HTML for styling porpuses. New content within the same class adapts the same style as revious content. The WebStorm project is also linked to GitHub as a backup solution. The tool used for uploading the website is filezilla. The project website is hosted on University of South-Eastern Norway (USN)'s server at https://itfag.usn.no/grupper/D13-25/.

6 Development Process

6.1 Methodology **OH |*TS***

Our product is a plug-in software for the existing infrastructure in KPCS. Because of this, an iterative approach would be the most optimal strategy. Iterating will make us able to respond to the feedback given by KPCS and makes the development process more effective. The project model for which we opted is the scrum model, the reason for this is because we wanted to match the KPCS workflow. They have a simplified scrum model where they do stand-ups 2 times a week, but we wanted a more comprehensive scrum model so we started with the daily stand-ups. This agreement was made early in the project to ensure consistent communication and to keep the team updated on progress, especially since we meet physically on a regular basis.

Defining scrum roles

Although Scrum is a relatively flexible framework, we aimed to adhere to its structure as accurately as possible at the beginning of the project. Through regular sprint retrospective meetings, we continuously evaluated and adjusted our approach based on what worked best for our team. As part of this process, we defined the classical Scrum roles to establish clear responsibilities. This resulted in less time being spent on reviews and retrospects both because they became more efficient and beacuse they felt like a waste of time.

- **Product owner: Tormod Smidesang** is responsible for communicating the product goal for the bachelor project. We see the bachelor project as the product. He is also responsible for setting the sprint goals each week.
- **Scrum master: Ole William Skistad Huslende** is responsible for making sure that the product owners goals are being meet. Also responsible for the scrum events throughout the week.
- **Developer team:** is responsible for the making of the individual tasks that needs to be done within the sprint, making sure that each task is a small subsection of the sprint goal. They also need to set estimated time on each of their tasks.

Typical work week

The typical week before and after Easter of KTT is shown in 5 and 6

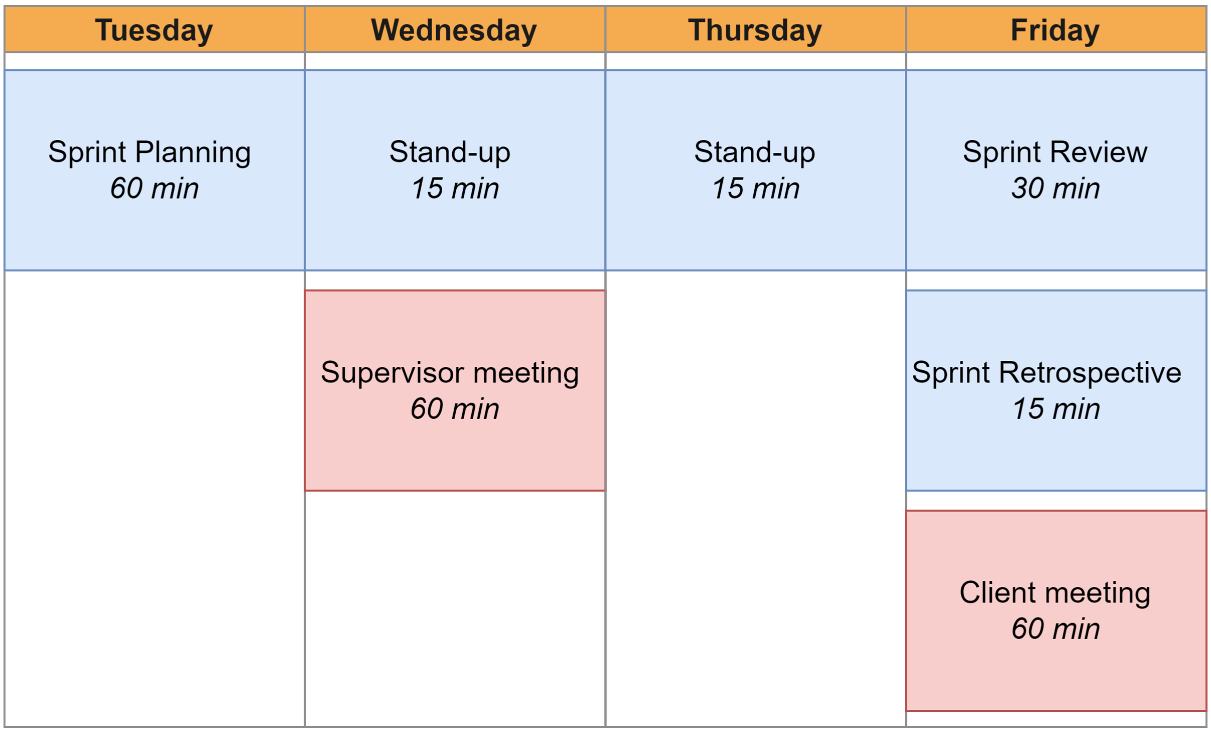


Figure 5: Scrum schedule **before** Easter

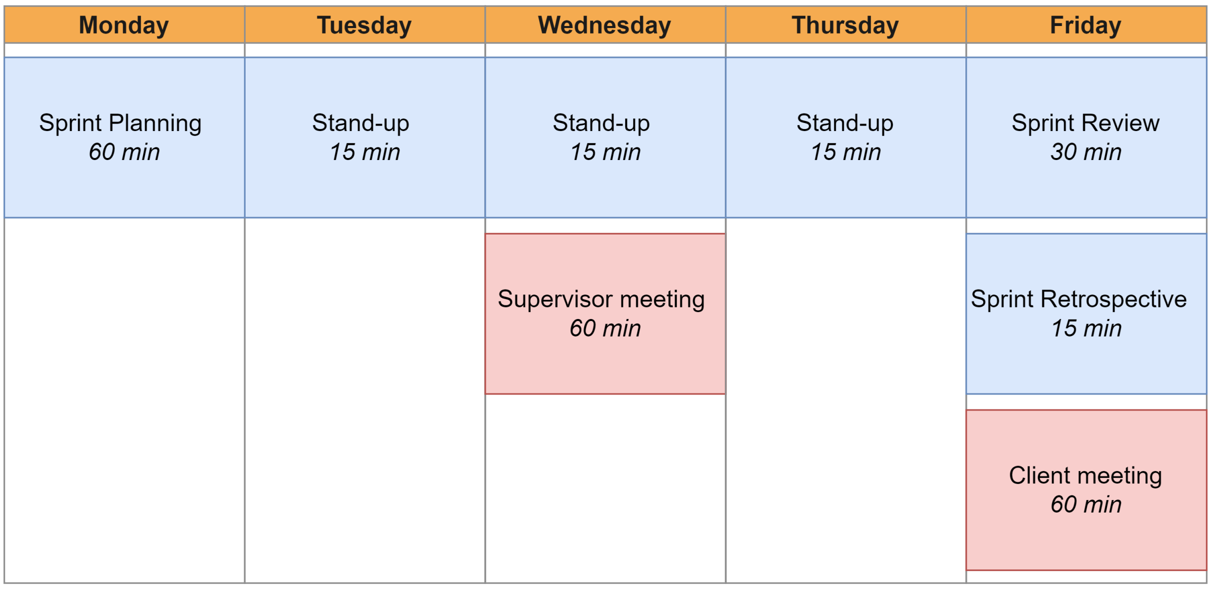


Figure 6: Scrum schedule **after** easter

These are the key components of our sprint week taken from [47]:

- **Sprint:** a time box that encompasses all the other Scrum events. In our case 1 week.
- **Backlog:** is a list of all the tasks for the whole project.
- **Sprint planning:** is a meeting with the developers, scrum master and project owner planning the tasks for the next sprint.

- **Stand-up:** is a meeting held every day where you walk through the three questions shown below:
  - What did I work on yesterday?
  - What am I working on today?
  - What issues are blocking me?
- **Sprint review:** is a meeting to summarize the previous sprint. We did this both with the external supervisor and internally. With the supervisor we show the progress that has been done and internally we update each other of the project state.
- **Sprint retrospective:** is a meeting with the entire scrum team talking about how to optimize and do the scrum model more effectively for our use.

6.2 Software Tools TS |*OH*

Programming language

There was some confusion in the beginning as to which programming language we should use. The initial document describing the assignment listed C# (with .NET MAUI) as an absolute requirement, while our external advisors said it was only a suggestion and that we could use whatever we wanted. Since one of the goals of our project is to create software that can be extended by the client afterwards, it makes sense to use the programming language they suggested. .NET MAUI is a framework for making cross-platform desktop and mobile apps in C# [48]

Visual Studio

Visual Studio is an IDE by Microsoft and has support for .NET MAUI. It is also the IDE we have used in previous courses so we are familiar with it and we all had it installed already. Visual Studio also has built-in support for GitHub and Copilot. [21]

GitHub

We set up a GitHub repository for the codebase and a repository for our LaTeX documentation. The client has requested the code base repository remains private for the time being, while the documentation repository will remain private because the final compiled Portable Document Format (PDF) document is all we need to present.

Jira OH |*TS*

We used the software tool Jira to make our scrum boards and to manage the backlog. This is the same software used in KPCS. In addition, we started to link git-commits with tasks in Jira. This is helpful for documentation, where we can look up a Jira task, and see which commit and what code implemented the solution to the task.

i-cut Production Console

i-cut Production Console (or iPC for short) is a software used to connect to and control KPCS cutting tables. The software includes Kongsberg HUB, which is the main interface our project interacts with and thus a key piece of software for us to be able to test our own software. See 2.2 for more information. [2]

Overleaf

Overleaf is an online LaTeX editor with support for collaboration for premium users. We have used Overleaf to write this document. Magnus purchased a premium plan so that we could collaborate together. The document was linked to GitHub as some of us have experienced connection problems with Overleaf before. That way, we will always have a copy of the report available. [49]

Doxygen

Doxygen is used to organize documentation written with comments in our codebase. It generates a pdf document that is included as an appendix in this report (Appendix K). Because we feel that comments make our codebase cluttered, we made a separate branch in our git repo only for writing comments for Doxygen.

ChatGPT/Copilot

We are using ChatGPT as a development tools. It helps write LaTeX, auto ceomplete code, and is helpful for questions along the way. It is important for us that ChatGPT is used in a sensible manner, where it only acts as a tool to help us in a way that does not take away key learning points of this project.

Draw.io

Draw.io is a free online diagram making software. We have used it to make various diagrams throughout the project. [50]

WebStorm

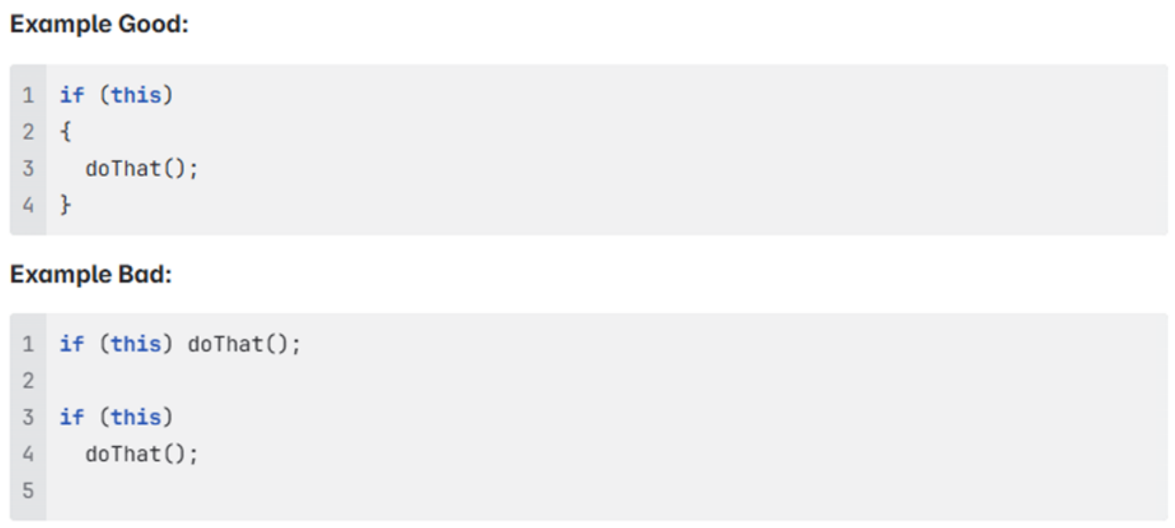
WebStorm is an IDE developed by JetBrains and was selected for website development primarily because it is easy to use. It also has several useful features such as code completion, error detection, and git integration. [51]

6.3 Coding Practices OH |*TS*

The product of this project is intended to be further developed and shipped by KPCS. To make a smoother transition, their coding standards must be implemented. There are some in-house rules, and the rest follows the Microsoft Framework Design Guidelines. The in-house rules consists of if statements, indenting and dependency injection. The naming conventions and the specific information can be looked up at [52]

Some of the rules from KPCS

Rule 1: Always use brackets in if statements.



Example Good:

```
if (this)
{
  doThat();
}
```

Example Bad:

```
if (this) doThat();

if (this)
  doThat();

```

Figure 7: Good and Bad If

Rationale: single line If statements complicates debugging, and brackets increases readability and clearly limits the scope of the statement.
Rule 2: Use four spaces instead of tab when indenting. Can be set up in Visual Studio to ensure correct indention.
Rationale: Readability suffers if not all indents are the same.
Rule 3: Resource Acquisition is Initialization
Stick to the constructor for dependency injection.

7 Requirements

This chapter explains the system requirements for the project. In the beginning of the semester, the company KPCS gave us the main goals for the system. These goals helped us understand what the system should do. From that, we created the system requirements explained in the following section.

7.1 System Requirements OH |*TS*

Below are the original project goals set by KPCS at the start of the semester.

1. Make a GUI to show the customer what happens their Kongsberg Cutting tables
2. Include job lists, current jobs and recently run jobs
3. GUI that can run on both PC and mobile device (cross platform)
4. Add and remove jobs from job list
5. Use .NET MAUI

Weekly meetings with KPCS quickly led to the creation of a list of user stories, derived from the original requirements. These user stories formed the basis for our initial draft of the system requirements. In collaboration with KPCS, we prioritized and ranked these requirements. As the project progressed, several requirements were revised, some were updated or replaced due to an evolving understanding of the system, while others were added based on new requests for functionality.

User stories shown in Appendix A are created based on the original project goals. They have the standard structure of **As a, I want, So that** and the acceptance criteria are structured with **Given, When, Then**. This is to clearly formulate what the user is requesting from the system. The user stories are further broken down into Use Cases, shown in Appendix B. Each of the use cases has one or more requirements attached to them. The requirements are clear goals that are central for the implementation process. In Appendix C is the complete overview from user stories, use cases and requirements. This shows a clear red line from user request to application design. The test method in Appendix H covers every requirement and if every test is passed the product is to be considered success.

Priority System OH |*TS*

The requirements have a priority ranking from A-C, and are explained in Table 4.

Priority	Description
A - Must Have (Critical/Required)	Essential requirements that must be implemented for the application to work
B - Should Have (Important)	These should be implemented but are not required for the system to work
C - Could Have (Nice-to-have)	These are more features that would be nice to have if there is time left

Table 4: Priority system

7.2 Test Implementation MO |*EP*

In this chapter, we will explain how we set up the different types of tests during the project. This includes unit tests, integration tests, system tests, acceptance tests, and regression tests. We describe how we wrote the tests, how we prepared the system for testing and why we did software testing.

Why test our code?

Testing was an important part of the project. It helped the team find bugs early and confirm that the application worked as expected. Because the system connects to physical machines and real-time data from the Kongsberg HUB, it was important to make sure that all components worked correctly together.

Testing gave confidence that the application would perform well in real use. For example, it helped check that table statuses were correct, that job data was displayed properly, and that buttons and features responded in the right way. Without testing, small errors could have caused wrong information, broken views, or connection problems.
Testing also made it easier to make changes during the development. After each update or fix, the tests helped check that the most important parts still worked. This was especially useful in a system that has both frontend and backend communication.

Software testing is known to improve reliability and reduce future problems. It supports a better user experience and lowers the risk of failure after delivery [53].

Test Strategy

As mentioned in the theory in Chapter 4.7, testing plays an important role in the development process.
The testing strategy for this project included several layers of testing to ensure the quality and stability of the system.

Unit Test Implementation

These tests were written using the xUnit testing framework in Visual Studio. We followed the Arrange–Act–Assert pattern to structure the tests in a clear and consistent way.
[54]
To test functions that communicate with the backend or external services, we used mocked dependencies. For example, we created a custom FakeHttpMessageHandler to simulate different HTTP responses without needing a real server. This allowed us to test API methods like GetActiveJobs, CheckAuthentication, and RequestAccessToken under controlled conditions. We also created fake services such as FakeApiService and modified service classes to accept injected HTTP clients, making it possible to isolate the logic being tested.

Unit tests were organized by separating them into folders based on layers (models, services, utilities), and further by class name. This made it easy to locate and run specific test scenarios. The tests covered multiple input cases per method using the [Fact] attribute to ensure behavior was correct for different types of data. Some methods were tested with valid and invalid inputs, including edge cases like null values and format mismatches.

While smaller helper methods were not tested, the tested areas focused on parts of the system that interact with job data, table logic, or authentication. These are critical for backend stability and catching logic bugs early.

This limited but focused unit test coverage was complemented by broader system and acceptance testing, which together helped ensure a stable and reliable application.

Integration Test Implementation

We tested how different parts of the code worked together. For example, we tested the connection between the user interface and the backend logic. The goal was to make sure that when something was done in the frontend, the correct part of the backend responded. This helped us see if the parts were connected in the right way and shared data correctly. We did not test everything at once, only how parts of the system talked to each other.

System Test Implementation

We also did system testing to check the full system from start to finish. This means we tested the whole application, not just parts of it. We tried to use the system like a real user would, to see if everything worked together as expected. This included testing different features, checking that the user interface showed the correct information, and making sure the system gave the right results. System testing helped us find problems that only happen when all parts of the system are running at the same time.

Acceptance Test Implementation

was done through a final meeting with KPCS, where the team presented the application and walked through the main features. Although the customer did not test the system directly, they had followed the development throughout the project and provided feedback along the way. During the session, the team demonstrated key functionalities and explained how the system meets the specified requirements. The walkthrough included navigating the interface, showing how jobs are uploaded and managed, and explaining how the application handles table communication and status updates. The goal of this session was to verify that the delivered system matched the expectations and requirements set earlier in the project.

Regression Testing

Regression testing was performed manually by the project group at the end of the development phase. The purpose was to confirm that recent changes and final updates had not introduced new errors. The team used structured test cases that were already created earlier in the project, based on user stories, use cases, and system requirements. These tests helped verify that all core functionality still worked as expected.

The group executed the tests step by step, checking that the system behaved correctly. For example, Test T-1 checked that a connection to the Kongsberg HUB could be established and that tables could be added using valid parameters. It verified that tables appeared in the GUI and no error messages were shown.
Test T-2 confirmed that table statuses were correctly displayed in the application and matched the actual state of the physical tables. Other tests confirmed that job data was shown and updated as expected.

This played an important role in making sure that the application was still stable after all final code changes. It gave the team confidence that the full system could be delivered without breaking existing features.

This structured testing approach made it possible to trace each test back to its requirement, and further to the related use case and original user story. The same trace could also be followed forward — from user story to use case, requirement, and finally to the specific test. This traceability improved the structure and quality of both the development and testing process.
As you can see in the table below, we can see which requirement the test is built upon and we can trace it all the way back to the user story. For the full regression test report see Appendix H

ID	Requirement	Test method	Test ID	Status
R-1.1.1	The application shall establish connection to the cutting table	Establish connection with the HUB	T-1	[Pass/Fail]

Table 5: Requirement and related test T-1

Regression test example

Example: Test T-1 – Establish connection and add tables

Step	Description	Comment
1	Open Kongsberg Hub Client Configurator on the Kongsberg Table.	
2	Press add table in the application and write the IP and Hostname.	
3	Write in the ClientID and the clientSecret.	The table should appear in the GUI with correct information, including name, status, image, and job data.
4	Click Confirm.	Table should now appear in the interface.

Table 6: Example of regression test T-1

Test coverage

The system was not fully covered by unit tests. Only the most important functions were tested, such as reading job data and checking table status. This helped the team find problems early in the backend, but smaller functions and helper methods were not tested. Other types of tests, like system tests and acceptance tests, were more complete. These tests followed a clear plan and tried to test all the main features of the application. The goal was to make sure that everything the user needed was working. The team tested the system step by step using real scenarios, so it was possible to find problems even in parts that were not covered by unit tests.
This combination of focused unit testing and broader manual testing helped make the system more stable and reliable.

8 Software Architecture

This chapter covers the architecture of the software program. It begins with going through the functional operations the user of the application are allowed to do. Then explaining the rationale behind the use of MVVM architecture pattern. Following this, the chapter details the core components of the system and how they interact. Keeping in mind how the data flows between the components. The section concludes with a diagram that shows the planned structure of the application.
This chapter outlines the key components of the design using MVVM architectural pattern. The application collects data from an external system called the Kongsberg HUB via RESTful API calls and SSE to provide the user with real time information. This chapter is intended to map out the main components of the system. The following section describes the responsibility of the model, view, view model and the supporting service layer.

8.1 High-Level System Specification OH | *TS*

The Kongsberg Table Tracker applications main objective is to let a table operator interact with the cutting table remotely as shown in Figure: 8. The operator interacts with the software to handle job lists and view requested statistics. All the data viewed is collected from the Kongsberg HUB. If the operator has multiple tables in the factory they should also be able to view them.

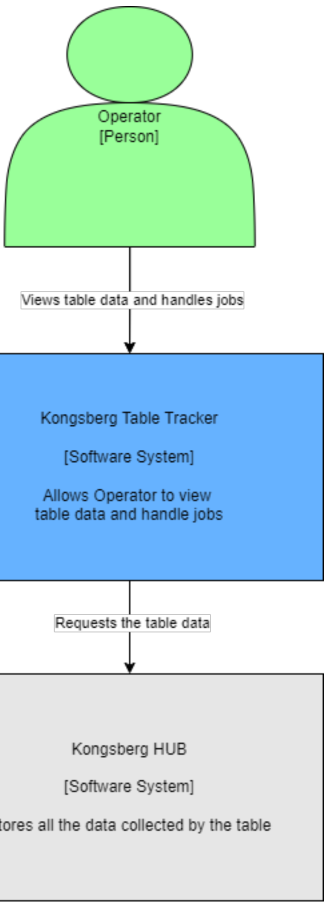

Operator
[Person]
Views table data and handles jobs
Kongsberg Table Tracker
[Software System]
Allows Operator to view table data and handle jobs
Requests the table data
Kongsberg HUB
[Software System]
Stores all the data collected by the table

Figure 8: Context diagram

Use case diagram of the application

The software system lets the operator add, delete and edit the information about the tables. This will make a list of tables where the operator is able to select a cutting table to get more information. Inside the single cutting table, the user should be able to manage the job lists as well look at information about them.

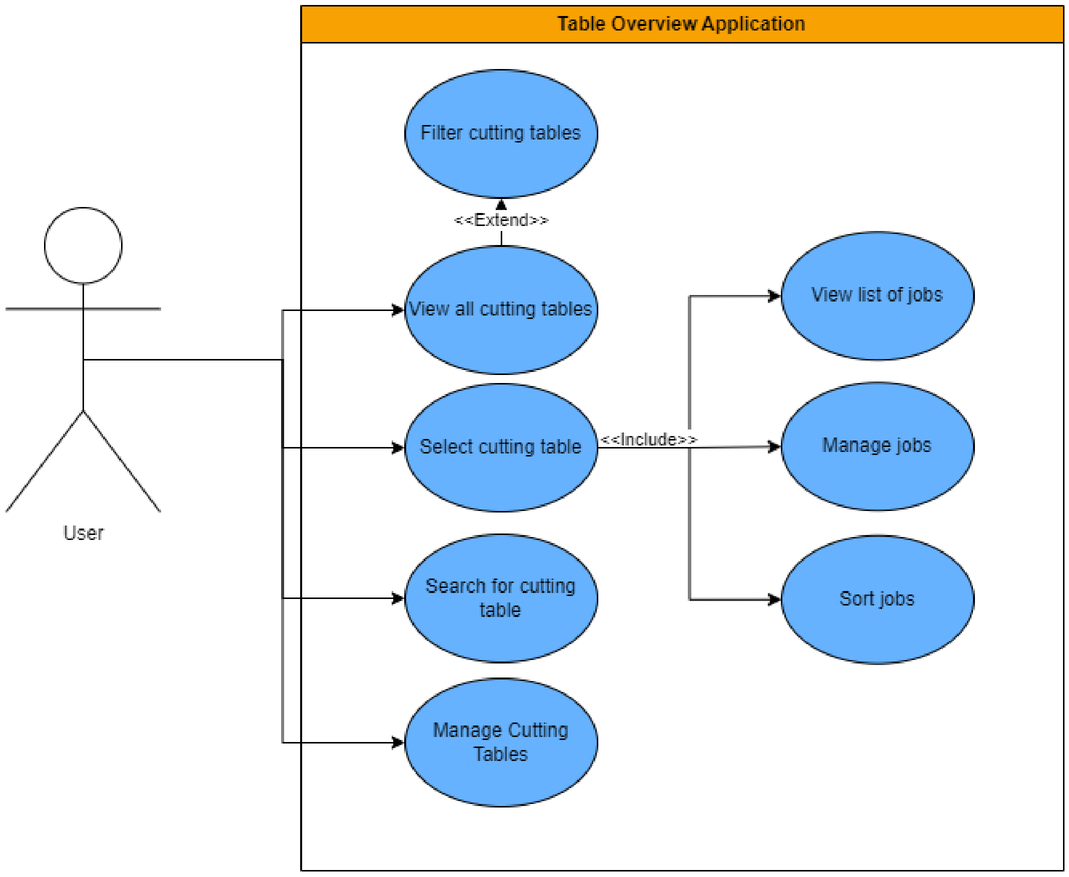


Figure 9: Use case diagram

8.2 Architecture

Choosing the Right Architecture MO |*TS*

In the early stages of the project, we needed to make a plan of how we would do the architecture. We had to look up different architecture patterns for the .NET MAUI framework. But before we got the time to do the reasearch, we had our first meeting with KPCS and they gave us some requirements to follow. This is because KPCS wants to build on our project after we hopefully have completed the base product, which naturally gave us more motivation for this project. One of these requirements was that we had to use the MVVM architecture when we coded. Since this architecture is the standard for .NET MAUI applications, we would naturally have come to the same conclusion ourselves.

Architecture pattern OH |*TS*

The MVVM architecture helps us maintain a clear separation of concerns between the business logic(Model), user interface (View), and interaction logic (ViewModel). This structure makes the codebase cleaner, more modular, and easier to manage. Additionally, it improves maintainability by decoupling the UI from the business logic, reducing the risk of errors when making changes to one part of the system.

Another key advantage of MVVM is that it makes testing easier. Since the business logic (ViewModel) is separate from the UI, it can be tested independently, allowing us to perform unit testing more effectively. This ensures that the core functionality of the application works correctly without relying on the user interface.

Furthermore, MVVM simplifies UI updates by allowing the ViewModel to act as a bridge between the Model and the View as shown in Figure: 10. This enables automatic updates of UI elements when data changes, making it easier to create a dynamic and responsive user interface. Lastly, the MVVM architecture supports scalability, allowing the project to grow and evolve without requiring major code refactoring.

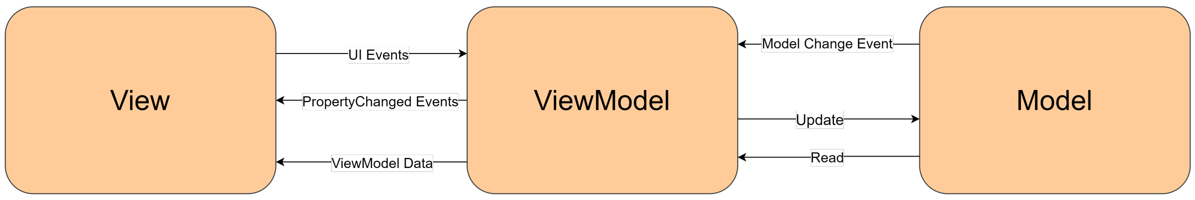


Figure 10: MVVM

8.3 Model **OH |*TS***

The model layer defines the data structure used throughout the application as shown in Figure: 18. These models represent the format of information exchanged between the external REST API and the system itself. Each model has properties that map directly to the JSON fields provided by the server. The primary model used in this application is the table model, which encapsulates a table. This class is not directly deserialized into but is used to hold all the model objects from the API calls. The lightweight classes used for deserialization are job, statistics, properties and trends. They represent the information from the API. They cover all the important aspects the Kongsberg HUB has to offer. The last important responsibility for the model layer is the event handler. The ApiService has the responsibility of handling the API calls as well as listening to events. The Kongsberg HUB sends out events via server sent events. They must be retrieved in the model and the model has responsibility to update itself so that the other structures in the application gets the updates.

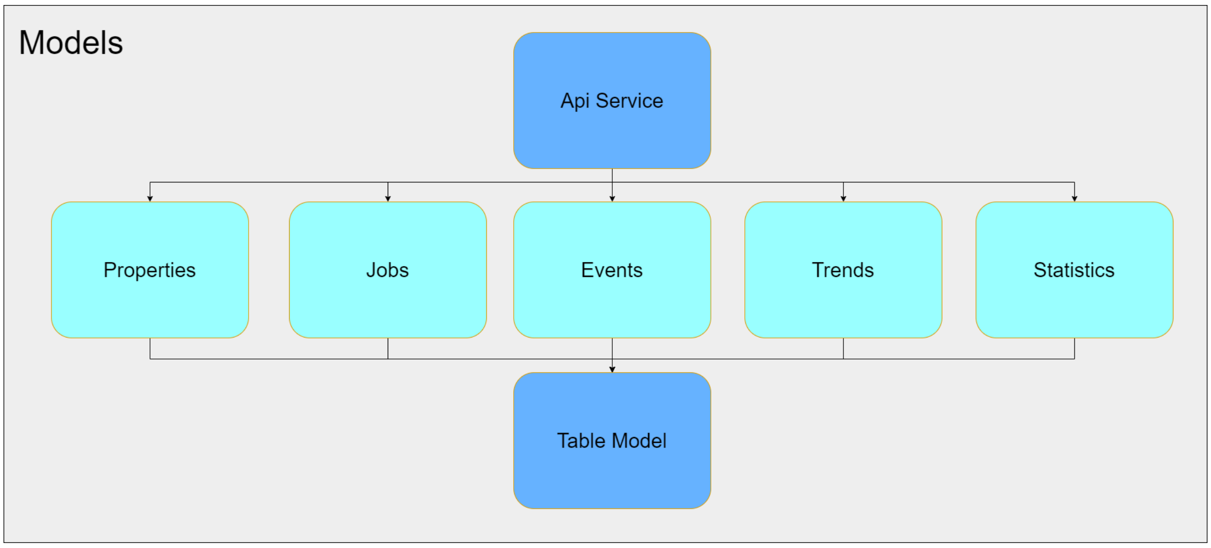


Figure 11: Models

8.4 View **EP |*MO***

The View represents the UI layer and is mostly written in XAML. The view does not contain business logic, since it should only focuses on the GUI and UX. By using the code-behind, it is possible to implement dependency injection and connect the view to a viewmodel. This is to enusre that the two conerns (GUI and business logic) are kept seperate.
In order to keep the GUI consisten, reusable and maintainable throughout the development process, it is important to use common resources. Two files that contribute to this are Colors.xaml and Styles.xaml. Specific colors are set and reused, ensuring the app maintain a consistant theme across different pages. Styles can help shape the controls used across pages. Margin, padding, background color, spacing, states, etc. are some of the properties that defines how each control will look. In the actual page, theese colors and styles are adopted through the Style property in the different controls.
The KTT application concists of several Pages and Popups that serve different purposes. Table 7 further explain what the purpose of each page is.

Page	**Purpose**
MainPage	Application entry point. Displays all tables.
StatusPage	New window of MainPage, but without functionality
TimeLinePage	Shows a list of tables and their jobs on a timeline.
MetaStatisticsPage	Display statistics gathered from all connected tables.
ActivePage	Display joblist for a single table
HistoryPage	Display job history for a single table
StatisticsPage	Display statistics for a single table
AddTablePopup	For adding tables
EditTablePopup	For editing existing tables
DiscoverTablePopup	Scan for tables on the same LAN
TableFilterPopup	Select filters for tables on MainPage
SettingsPopup	Select language and delete all table credentials
ColumnSelectiorPopup	Select categories from job list that are displayed
CustomStatisticsPopup	Select a custom time span to collect statistics from

Table 7: List all pages & popups

Navigation is also an essential part of the UX. The navigational strucutre for the KTT application is explained in Figure 12.

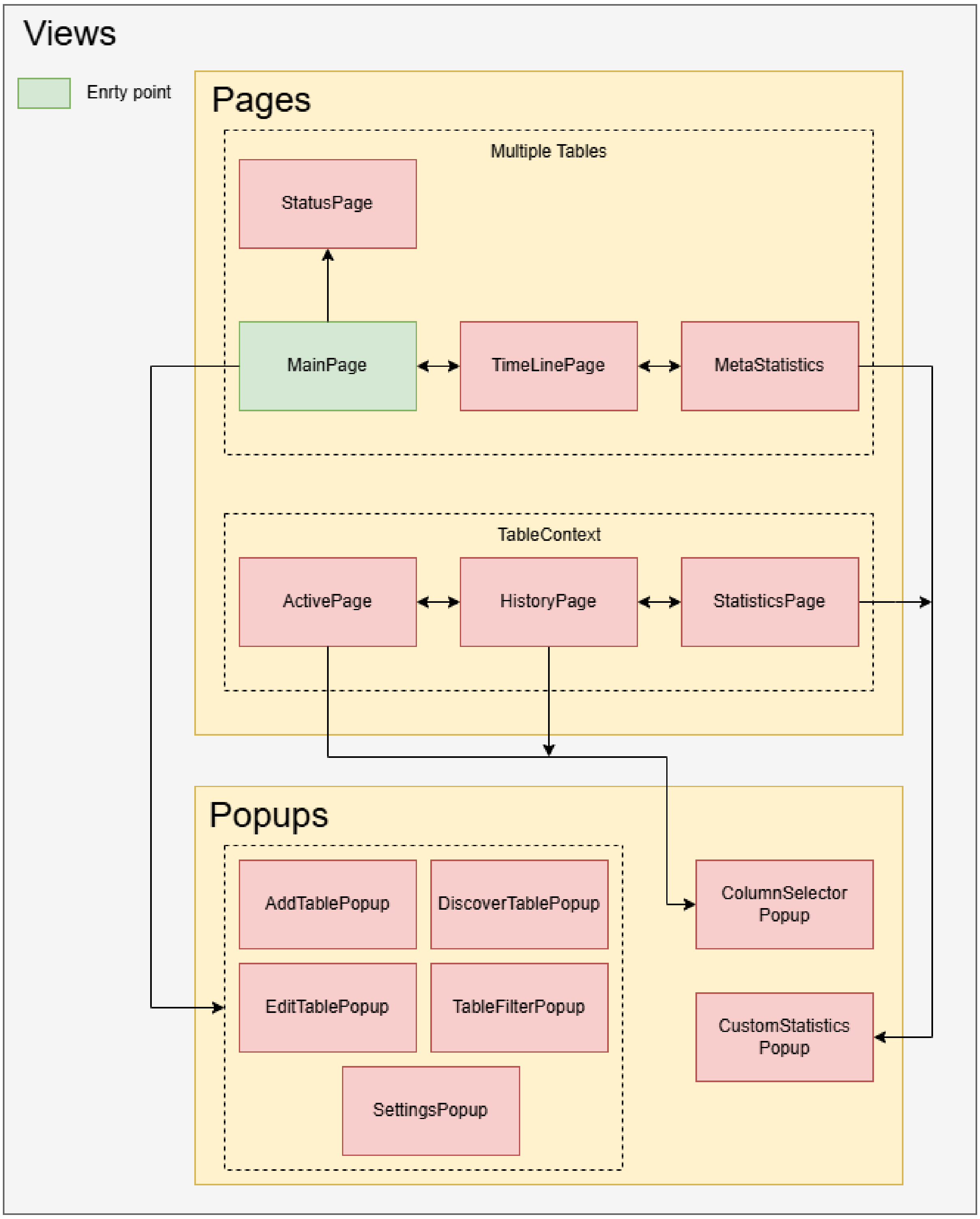


Figure 12: Navigation diagram

Layouts of the pages are important because they have a significant impact on the UX. A well designed layout lets the user navigate the application intuitively, without the need of prior knowledge. A layout includes elements such as navigation structure, spacing between components and positioning of control. A good layout stays consistent to avoid confusion, and helps guide the users attention to the important parts of the page. Figure 13 shows a mockup of MainPage in the KTT application. The top bar is responsible for navigation, and works like a tab-bar. The main content of the page is the tables in the middle of the mockup. Each element represents a different table, and shows that by displaying table name, active job and current status of the table. See Appendix E for a collection of mockups for all the content pages in the application.

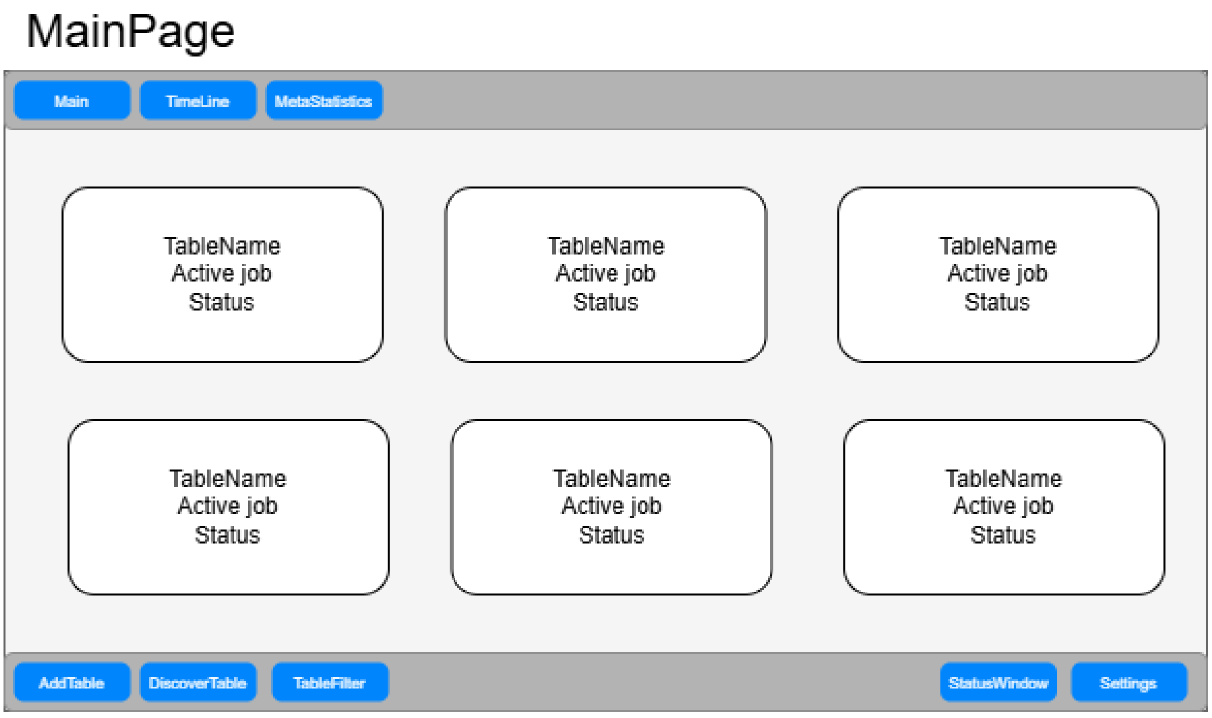


Figure 13: MainPage mockup created in draw.io

8.5 ViewModel **OH |*TS***

The viewmodel layer defines the intermediary between the model and the view. The viewmodel layer encapsulates the applications presentation logic as well as holding the data for each of the views. It exposes data and commands to the view, so the view can interact with the model. The viewmodels should be connected to the pages, and it should get the appropriate data from the model layer. In the design, the viewmodel layer consists of the table context which is the viewmodels corresponding to a single table and the external viewmodels that holds the data of all the connected tables. The main components in the table context as shown in 14 are the active jobs, history jobs, statistics and preview. The active jobs holds the current job list on the table so the view can display it. It also has the logic for interacting with the jobs, like adding and deleting. The history jobs are the list of the jobs that have been run. The statistics data are the history of everything from information about the table, what a customer has ordered, what material has been used and so on. This has to be extracted from the data in the model. Then you have the preview this is the data that gets exposed to show essential data of the table. The external viewmodels as shown in 14 consist of the main, timeline and meta statistics. The main previews and handles all the tables and lets you add, edit and remove this is done by commands that the view binds to. The timeline holds a list of all the active jobs for each table, so you can se how production will continue. Lastly there are the meta statistics that merge the important statistics from each table, essentially holding the statistics for the whole factory.

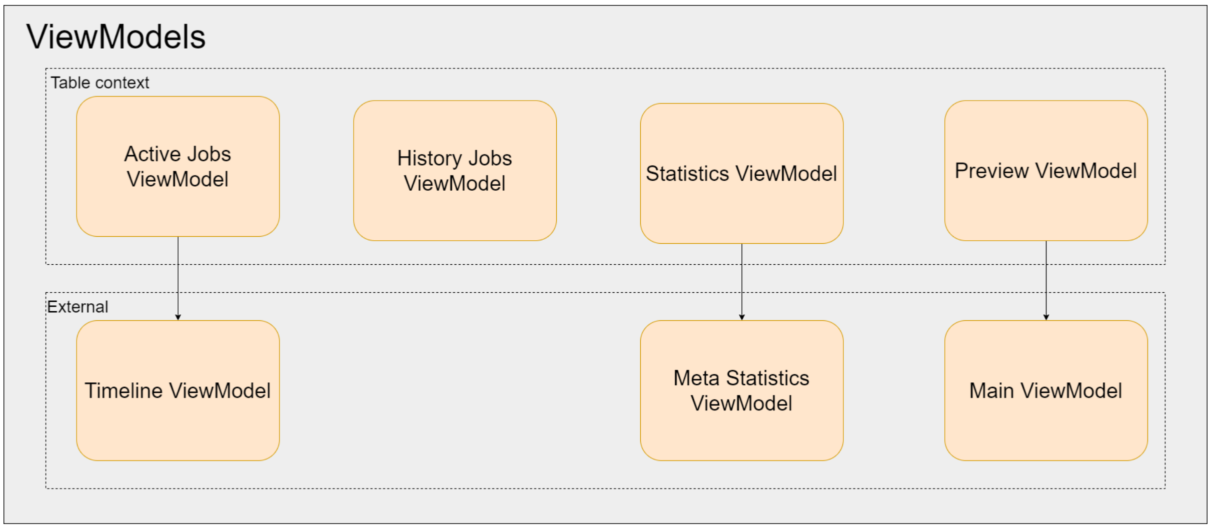


Figure 14: ViewModels

8.6 Services **OH |*TS***

The application must support multiple tables. The table context is the collective name for the model and viewmodels for the single table. All the components of a table are instantiated in a single table context component, like a singleton. The TableContext service is the service that handles multiple tables and it should expose the data from the table contexts to the external viewmodels. It is the intermediary between the single table viewmodels and the multiple table viewmodels. The tables should not disappear when you restart the applications so the service is also responsible for saving and loading the tables.

8.7 Data Flow **OH |*TS***

The data flow of the system follows a **Model-View-ViewModel (MVVM)** architecture, ensuring a clean separation of concerns and a scalable design. Figure: 15 illustrates the interaction and messaging structure within the system.

Messenger Buses

The application utilizes two primary messenger buses:

1. **External Messenger:**
   - Facilitates communication across the entire application.
   - Connects the single table with viewmodels using data from multiple tables.
2. **Table Context Messenger:**
   - Dedicated to handling messages related specifically to the table context.
   - Connects the Table Context Model, ViewModel, and UI Page in a localized, modular way.

Data Flow Overview

1. **Kongsberg HUB:**
   - Acts as the central data source, responsible for both sending and receiving data.
   - Sends data to the appropriate Table Context Model based on incoming requests or commands.
   - Can receive messages back from the UI layer.
2. **Table Context Model:**
   - Processes incoming data from the Kongsberg HUB.
   - Propagates the data to the Table Context ViewModel.
   - Publishes messages through the Table Context Messenger for localized updates.
3. **Table Context ViewModel:**

- Receives data from the model and manages its presentation logic.
- Sends data change messages to update the UI.
- Acts as a mediator between the model and the view.

4. **Table Context Page:**
   - Observes and reflects changes triggered by the ViewModel.
   - Sends user commands back through the messaging chain.
5. **Tables ViewModels and Pages (Second Layer):**
   - Communicate with the individual table context via the External Messenger.

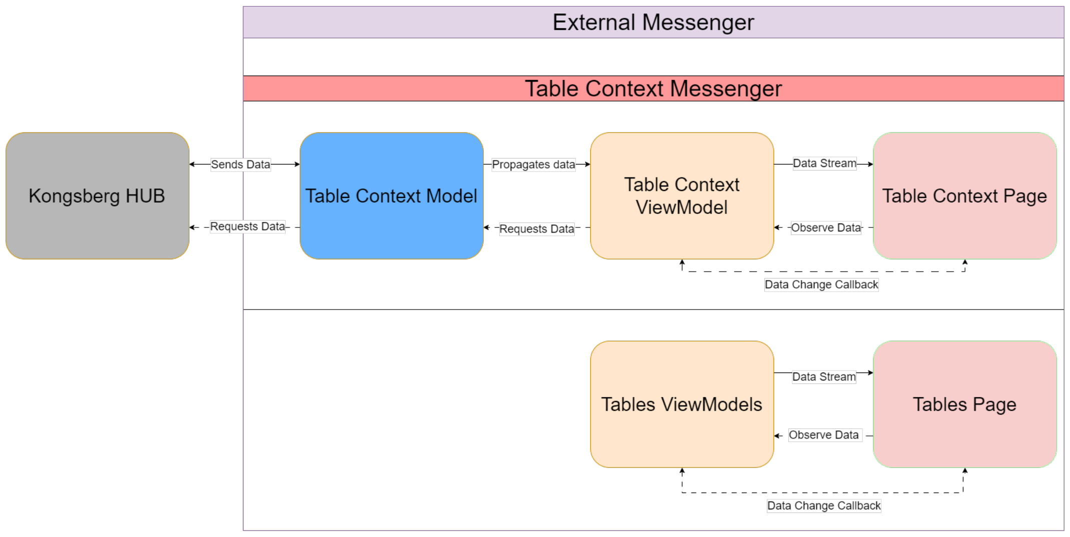


Figure 15: MVVM dataflow

8.8 Architecture Diagram OH | *TS*

The final iteration of our design is shown in Figure: 16. This diagram clearly illustrates the data flow throughout the application. It starts at the Kongsberg HUB, where all data is stored. Both the ApiService and the table model represent core parts of the MVVM structure.
Data is first retrieved by the ApiService and then populated into the table model, which in turn represents a single table. This table model transfers the appropriate data into the corresponding viewmodels. One viewmodel, for example, manages the active jobs for that particular table. It allows users to add or remove jobs. Additionally, the model sends data to the history job module, which shows the produced jobs.
The design incorporates a statistics module that aggregates table specific statistics. The preview viewmodel acts as the central component for the main page, displaying the relevant information for that spesific table. There is a clear layering within the table context that includes all necessary models, views, and viewmodels to represent a single table.

A separate table context service is responsible for managing all table contexts, thereby aiding the meta-statistics viewmodel, timeline viewmodel, and main viewmodel in accessing table data. The meta statistics module aggregates data across all tables, while the timeline viewmodel provides an visualization of upcoming active jobs. The main page serves as the primary entry point of the application, and the status panel is a simplified version of the main page. Displaying the current status of all the connected tables. With this architecture we make a scalabe and thought out application that will cover all the requirements.

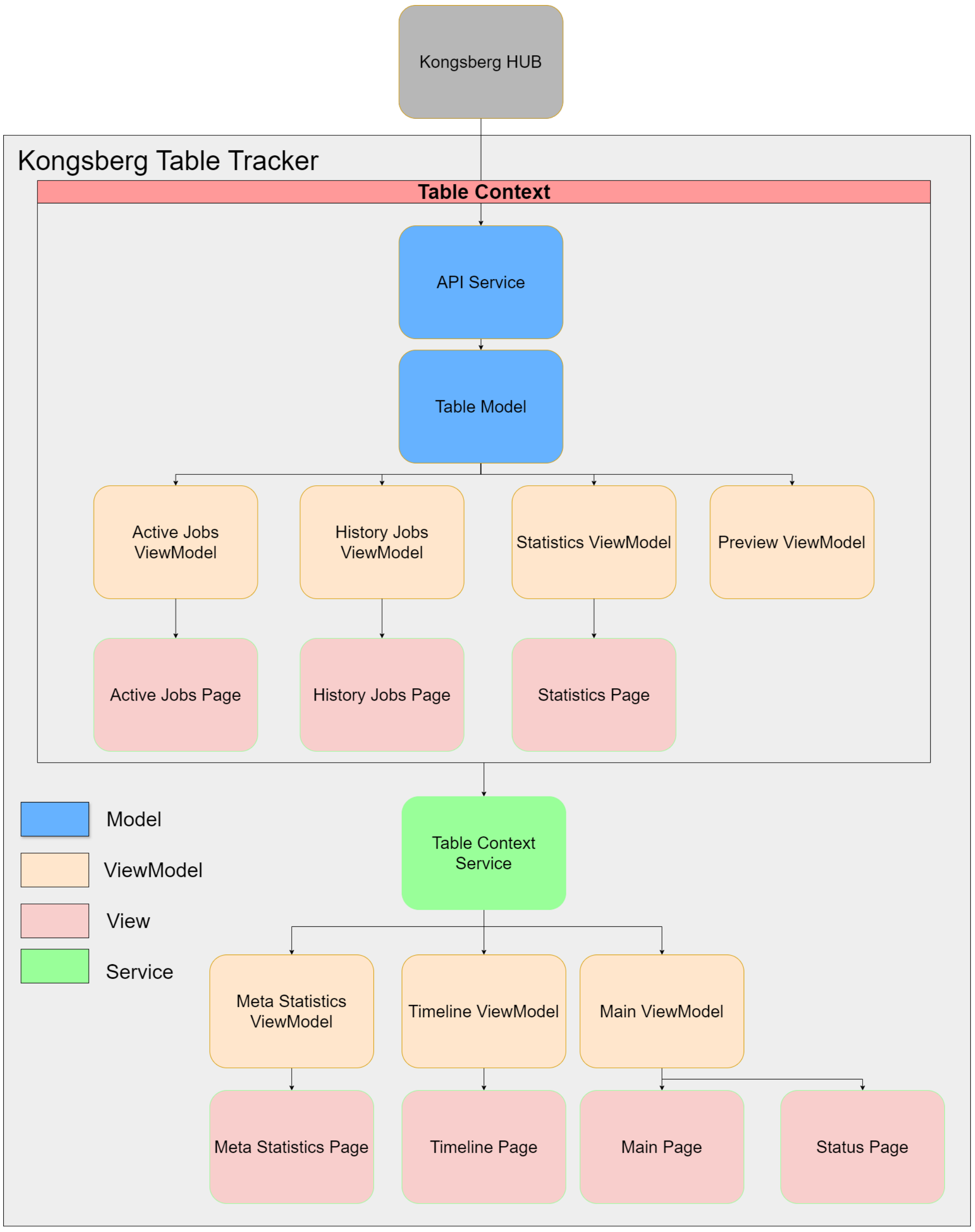


Figure 16: Architectural diagram

9 Implementation

This chapter is about the implementaion of what was designed in the previous chapter. It covers important aspects of how the team worked towards actually building the product. It is organized in a logical manner, starting with the developer environment, traceability, and then a bottom up approach for the application.

9.1 Development Environment

Folder structure **TS |**

```
/
├── Models
│   ├── ApiProperties
│   │   └── Enums
│   ├── ApiQueue
│   │   └── Enums
│   ├── ApiSSE
│   ├── ApiStatistics
│   │   └── Enums
│   └── ApiTrends
├── Services
├── Utilities
│   └── Converters
├── ViewModels
│   ├── Popups
│   └── TableContextViewModels
│       └── SubViewModels
└── Views
    ├── Components
    ├── Popups
    └── TableContextViews
```

External libraries **EP |*TS***

One of the most important external libraries included in the KTT application is the CommunityToolKit.Mvvm by Microsoft. It drastically reduces boilerplate and improves code clarity. It is no longer necessary to write INotifyPropertyChange, ICommand and PropertyChanged notification. Instead, it is possible to use [ObservableProperty] in the ViewModel. When the View is bound to this property, it automatically updates when a change occurs. Another useful tool is the [RelayCommand], which does the same as the [ObservableProperty], but for functions instead of variables.

Local installation of iPC **TS |*OH***

Because our application has to rely on the presence of a Kongsberg HUB, we received license keys to iPC from KPCS so that we could install it on our own laptops. Unfortunately, we are not able to install iPC on any laptops running a Home Edition of Windows. This means that only half of our group has access to a Kongsberg HUB on their own machine.

Network setup **TS |*OH***

If a local installation of iPC is not available, we need a computer network that lets us talk to a Kongsberg HUB. Over at KPCS, their local network has numerous cutting tables available, each running an instance of the Kongsberg HUB. However, when we are working from somewhere else, we have no cutting tables available. For example, the network configuration on the university Wi-Fi (eduroam) prevents us from accessing services on other computers connected to the same Wi-Fi. For this reason, we have a network switch in our room at Bergseminaret that we connect our laptops to with ethernet cables. This way, even though half the group are unable to install iPC on their laptops, they still have access to a Kongsberg HUB through the network whether we are in Bergseminaret or at KPCS

9.2 Traceability TS |*OH*

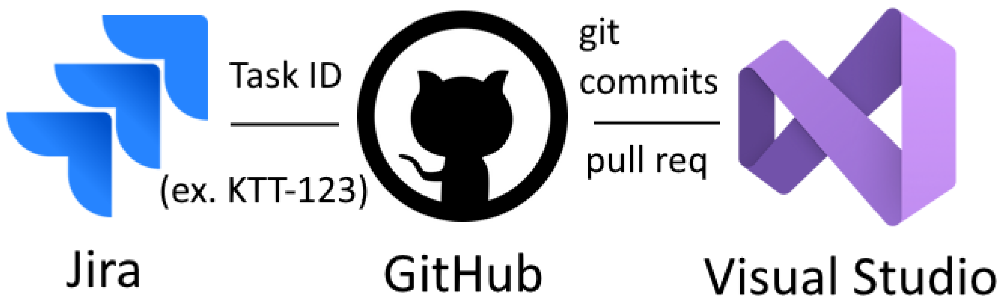


Figure 17: Traceability

In order to work in an efficient and organized manner, we put our user stories, use cases and derived requirements into Jira and linked them together. From there, we made specific code implementation tasks and linked them to the relevant requirements. Tasks are automatically assigned an identifier in the format KTT-xxx, where xxx is a number. We linked our Jira project to our GitHub repository so that we could make git branches directly from Jira. These branches are then given names that begin with the task identifier. By ensuring our branches, commits and pull requests all begin with this task identifier, Jira will keep track of them so that we can see which commit in the repository implemented which requirement. Git also integrates nicely with Visual Studio so we can right click somewhere in the code and select Git, then Blame to see which commit introduced which line of code. From there, we can see the task identifier, leading us to the connected task. Since the task is linked to a requirement, we can see a clear line all the way from a user story down to each line of code used to implement it and vice versa. A table showing the tasks and which requirement they are linked to can be found in Appendix D

9.3 Model OH |*TS*

The model section talks about how the model layer was implemented in software architecture. In Figure: 18 you see the implementation structure of the layer. It includes the Models, the ApiService and the table model. Each of them is explained in the next sections.

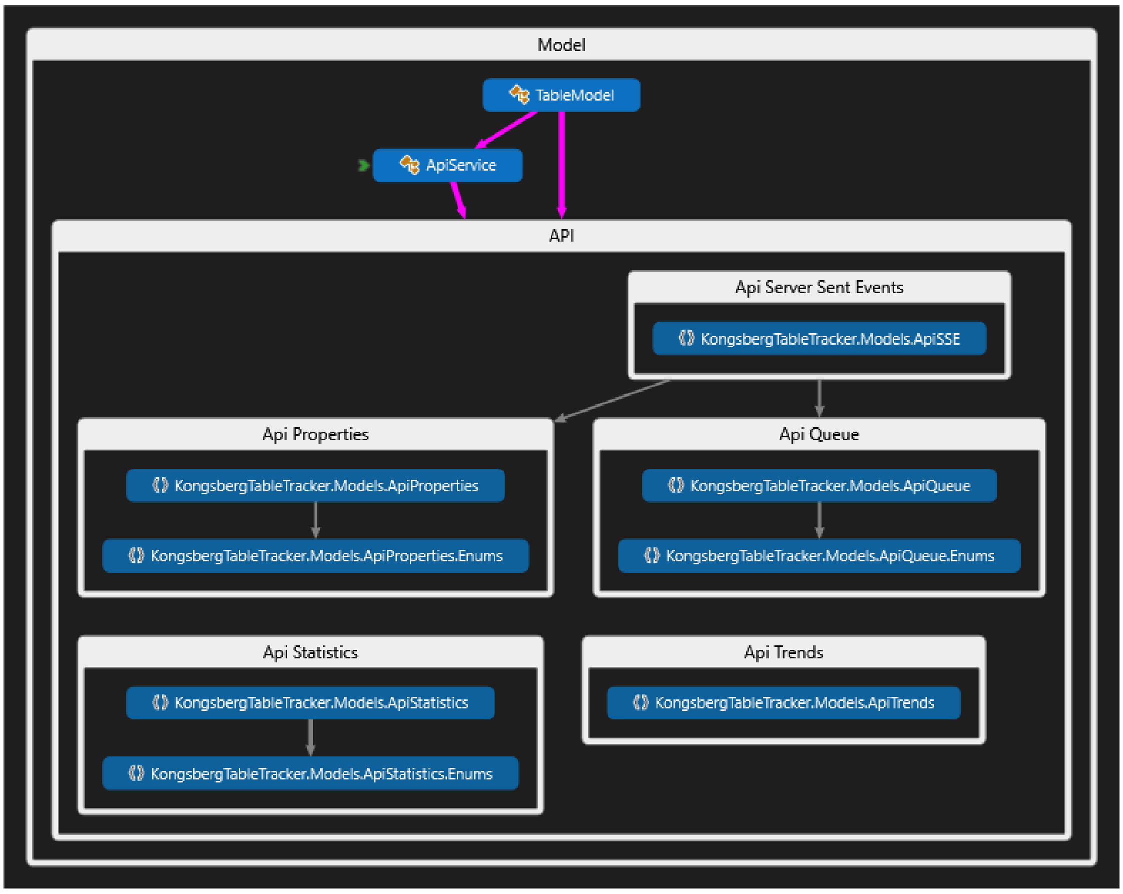

Model
TableModel
ApiService
API
Api Server Sent Events
KongsbergTableTracker.Models.ApiSSE
Api Properties
KongsbergTableTracker.Models.ApiProperties
KongsbergTableTracker.Models.ApiProperties.Enums
Api Queue
KongsbergTableTracker.Models.ApiQueue
KongsbergTableTracker.Models.ApiQueue.Enums
Api Statistics
KongsbergTableTracker.Models.ApiStatistics
KongsbergTableTracker.Models.ApiStatistics.Enums
Api Trends
KongsbergTableTracker.Models.ApiTrends

Figure 18: Model

Models

KPCS has defined classes that they send as responses to HTTP requests. Each of the endpoints has specific classes implemented. They get delivered in JSON format and the standard .NET deserializer inputs the content into the models. There are some Enums that need to use its own serializer options to input the correct data into. This is mainly to get a shorter name of the Enum value. The models hold all the data from the endpoint. The data is separated into 5 main categories jobs queue, history job queue, table properties, table statistics and property trends. Their properties and explanations are all shown in Appendix K.

ApiService

ApiService is the bottom layer of our application. It is the service that connects to the Kongsberg HUB. In .NET MAUI this is done through the HttpClient class. All the properties and methods are shown in Figure: 19. A new ApiService is added with every new table, it configures its own HttpClient and the correct base address. The base address is the "https://ip + path". When the HttpClient is configured it has to get an authentication token from the Kongsberg HUB. This is done by sending a POST request with the clientId and clientSecret received from the user. The response is then included as a bearer token in the HttpClient so that every subsequent request made by it will have the token in the package. Then it sets up the event listener and gathers all the data from the Kongsberg HUB, from almost every endpoint.

The endpoints are not disclosed in the report by request from KPCS, but they can be found in the code.

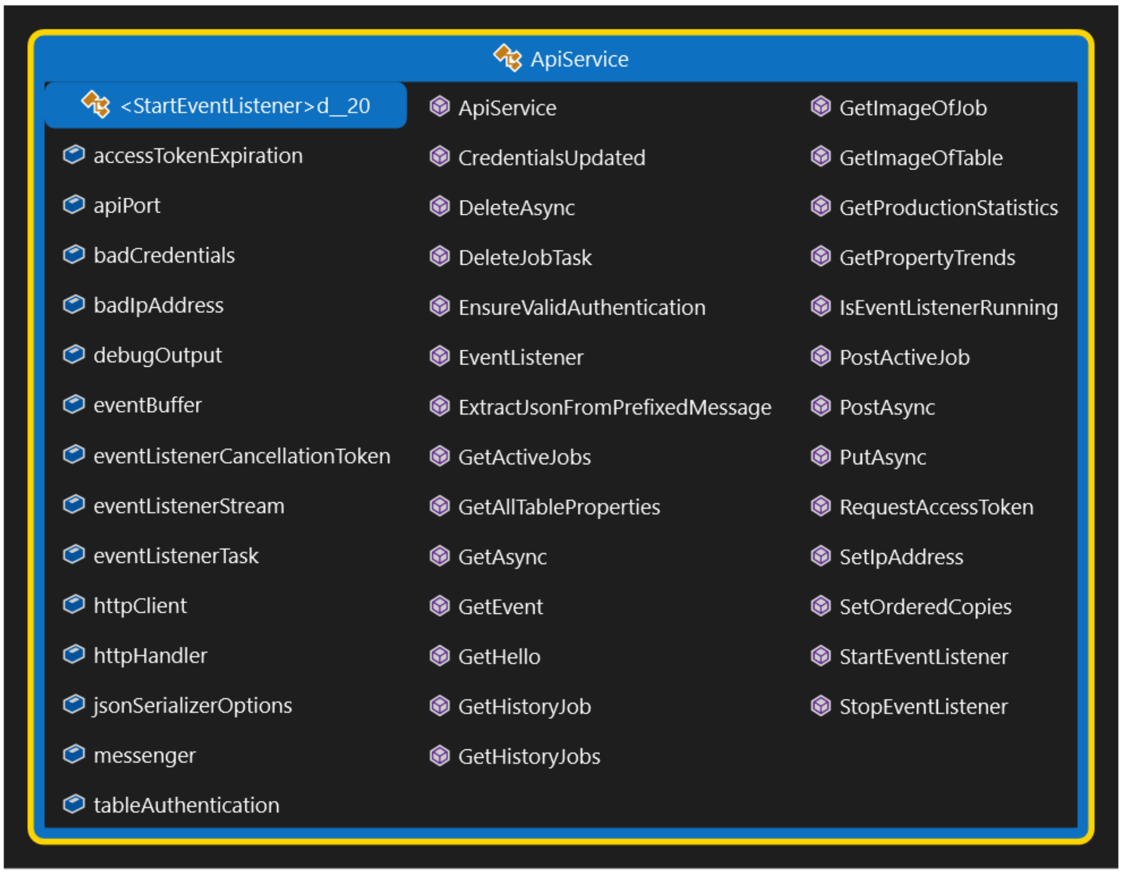


Figure 19: API Serivce

Authentication
The user inputs the clientId and clientSecret generated by the Kongsberg Hub Client Configurator application made by KPCS. The credentials are then sent back into the stack to the ApiService class. The class sends a POST request to the Kongsberg HUB and receives an authentication token. This token is saved in an authentication class and used in the HttpClient class. The authentication class also has a property that records the time the token was set. Each time you make a request via the HttpClient it checks the time and if it's over 14 days, as set by the Kongsberg HUB, it will then ask for a new access token.

Event Listener
The event listener is shown in Listing: 1. It is run on a separate thread that is called a Task in .NET. The Task is given a cancellation token that can be used to stop the task when necessary. The StreamReader class from .NET subscribes to the SSE from the Kongsberg HUB. The StreamReader is setup in a StartEventListener function. The Kongsberg HUB sends out the event in the format of a stream. The "event description" is filtered out because in the current Kongsberg HUB version, it does not contain any important information. The rest of the stream gets formatted and checked to see if its valid JSON format. The events get enqueued to the event buffer that the event handler can read from.

Listing 1: Event Listener

```
public async Task EventListener(CancellationToken token)
{
    Debug.WriteLineIf(debugOutput, $"EventListener() called");
    Debug.WriteLineIf(token.IsCancellationRequested,
        $"attempted to begin listening for events on
        {httpClient.BaseAddress?.Host}, " +
        $"but the cancellationtoken was already set to cancel");
    while (!token.IsCancellationRequested)
    {
        try
        {
            while (!eventListenerStream.EndOfStream &&
            !token.IsCancellationRequested)
            {
                string? response = await
                eventListenerStream.ReadLineAsync(token);
                if (!string.IsNullOrEmpty(response) &&
                !response.Contains("event: DeviceUpdated"))
                {
                    var json = ExtractJsonFromPrefixedMessage(response);
                    eventBuffer.Enqueue(json);
                }
            }
        }
        catch (TaskCanceledException)
        {
            Debug.WriteLineIf(debugOutput, $"EventListener():
            TaskCanceledException caught" +
                $"\n\tcancellationtoken.iscancellationrequested:
                {token.IsCancellationRequested}");
        }
        catch (Exception ex)
        {
            Debug.WriteLineIf(debugOutput, $"ApiService.EventListener():
            caught exception: {ex.GetType()}");
            if (!token.IsCancellationRequested)
            {
                Debug.WriteLineIf(debugOutput, $"Error in
                ApiService.EventListener()
                ({httpClient.BaseAddress?.Host}): {ex.GetType()}\n\tTrying
                again in 5 seconds");
                await Task.Delay(TimeSpan.FromSeconds(5));
            }
        }
    }
    Debug.WriteLineIf(debugOutput, "EventListener finished executing");
}
```

Data Loading
The other main component of the ApiService is the data loading aspect. The ApiService has template functions for GET, POST, PUSH and DELETE. The functions are used in the single function for each of the endpoints. They return the deserialized data model to the table model. The main data areas is the table job queue, history job queue, table properties, property trends and statistics. The table model class requests an initialization and the API serves it.

Table Model

The main model used though the application is the Table Model. The class holds all the information of that table. The credentials are saved in the model. It holds an ApiService as well as all the models returned from ApiService. Its main functionality is the data initialization and the event handler.

Initialization
When the table is made in the view and credentials are propagated back to the model you instantiate the table model with the ApiService inside. The model then requests all the data by calling the functions in ApiService. They return the models deserialized from each of the endpoints.

Event Handler
The ApiService is then connected to the event stream from the hub and fills a queue with each of the events. This queue is a queue of JSON string corresponding with each event. The table model has a task running on a separate thread the code is shown in the code segment 2. That task gets an event from the service and finds the type of event it is. Then call that event specific function. The functions updates the table model and sends out messages to the viewmodels listening.

Listing 2: Event Handler

```
 private async Task HandleEvents(CancellationToken token)
{
    Debug.WriteLineIf(debugOutput, $"TableModel.HandleEvents(): starting
    event handler");
    Debug.WriteLineIf(token.IsCancellationRequested, $"Error in
    TableModel.HandleEvents():\n" +
        $"\tattempted to start event handler for {Name}, but the token was
        already set to cancel");
    while (!token.IsCancellationRequested)
    {
        string eventData = apiService.GetEvent();
        if (!string.IsNullOrEmpty(eventData))
        {
            JsonDocument jsonEvent = JsonDocument.Parse(eventData);
            JsonElement root = jsonEvent.RootElement;
            if (root.TryGetProperty("Action", out JsonElement elementValue))
            {
                string? eventAction = elementValue.GetString();
                if (Enum.TryParse(eventAction, out EventAction eventType))
                {
                    switch (eventType)
                    {
                        case EventAction.TablePropertiesChanged:
                            OnNewTablePropertyReceived(eventData);
                            break;
                        case EventAction.QueueEntryAdded:
                            OnQueueEntryAddedReceived(eventData);
                            break;
                        case EventAction.QueueEntryUpdated:
                            OnQueueEntryUpdatedReceived(eventData);
                            break;
                        case EventAction.QueueEntryMoved:
                            OnQueueEntryMovedReceived(eventData);
                            break;
                        case EventAction.QueueEntryRemoved:
                            OnQueueEntryRemovedReceived(eventData);
                            break;
                        case EventAction.HistoryEntryAdded:
                            OnHistoryEntryAddedReceived(eventData);
                            break;
                        case EventAction.HistoryEntryRemoved:
                            OnHistoryEntryRemovedReceived(eventData);
                            break;
                        case EventAction.EventLogAdded:
                            break;
                        default:
                            Debug.WriteLineIf(debugOutput, $"Unknown event
                            type: {eventAction}");
                            break;
                    }
                }
            }
        }
        else
        {
            await Task.Delay(100, token);
        }
    }
    Debug.WriteLineIf(debugOutput, $"TableModel.HandleEvents(): stopping event handler for {Name}");
}
```

9.4 View EP |*MO*

This section describes the view layer of the KTT application. Each page consists of two parts: a XAML file that contains the visuals of the page, and a code-behind that is responsible for initialization and connection to the business logic behind. The following subsections provide an overview of the Main Page in the application. Figure 27 is a screenshot with annotation for different elements on MainPage. Following that is a section of the page structure/layout, and relevant code. It is worth mentioning that icons found in the applications are either provided by KPCS, or found in the free Font Awesome library. See licence [55]. For a more detailed explanation of all the different pages, see Appendix F

Overview of MainPage

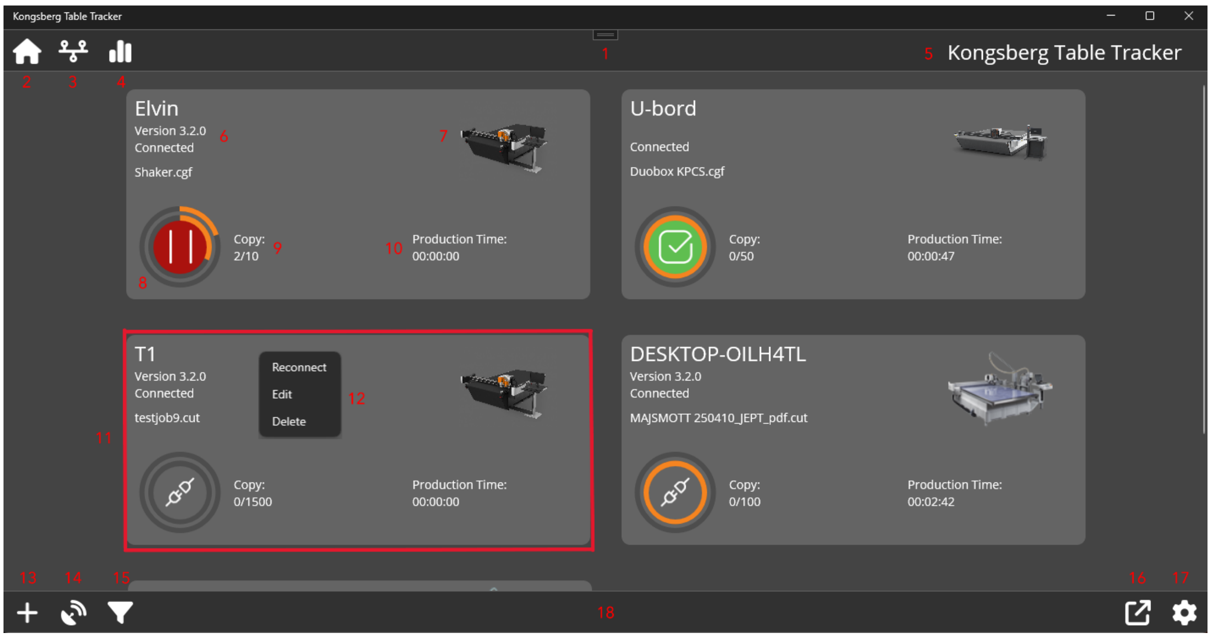


Figure 20: MainPage screenshot with annotated UI elements

1. **Navigation bar** - Container for navigation-buttons and page detail.
2. **Home button** – Navigate to MainPage when pressed.

3. **Timeline button** – Navigate to TimeLinePage when pressed.
4. **Meta Statistics button** – Navigates to MetaStatisticsPage when pressed.
5. **Page detail label** - On main page, it displays application name.
6. **Table details** - Table name, iPC version, connection and job name.
7. **Table Image** - Image of table model.
8. **Satus** - Display status for table and current job.
9. **Copies** - Completed copies and total ordered copies.
10. **Production time** - Production time of the job.
11. **Table item** - Collection item of a specific table.
12. **Right click options** - Right click brings up a selection menu.
13. **Add table button** - Open popup for manual entry off a table.
14. **Discover table button** - Open popup and scan selected interface for tables.
15. **Table filter button** - Filter tables without credentials.
16. **Status button** - Open new window of main page, without functionality.
17. **Settings button** - Open popup for application settings.
18. **Bottom bar** - Container for main page buttons.

Structure

As mentioned in the Theory chapter, a ContentPage can only hold one piece of content. On MainPage, that content is a Grid Container, which further divides the page into three rows. The first row (Grid.Row=0) is a custom navigation bar(4) and is of type ContentView. The NavBarContentView is defined in a separate XAML file, and injected into the MainPage. This makes the navigation bar more reusable and maintainable for further development. The navigation bar contains three different hover:buttons(1,2,3) that each navigates to a different page. A hover:button is a custom class that is derived from the ImageButton class, and includes animations for when the pointer is hovering over the button icon. All buttons also have tooltips so that the user can get a more detailed description, other than the icon. The NavBar also includes a page description label(5), to inform the user of which page they are on. The navigation bar works like a regular tab bar and stays consistent while navigating between MainPage, TimeLinePage and MetaStatisticsPage.

The next row (Grid.Row=1) is also of type ContentView. This ContentView takes up the main section of the window, and is called TablesContentView. Just like the navigation bar, it is also injected from a separate XAML file. The reason TablesContentView is made this way, is because it is used on both the MainPage and StatusPage. It contains a CollectionView of all the tables that have been added by the user. Each table item (6) displays a table with name, iPC version, connection status, table status, job status, job copies, image of the table and production time. It also has a right click option (MenuFlyout) that lets the user reconnect, edit or delete a table from the CollectionView (7). By left-clicking the table, the user will navigate to the ActivePage of that specific table. The navigation bar will change to display navigation buttons for specific table related pages (ActivePage, HistoryPage and StatisticsPage).

The last row (Grid.Row=2) is the bottom bar(13), and is a Grid wrapped in a Frame. It contains five different buttons. The first button (8) opens a popup for manually adding tables. A table item is added to the CollectionView when confirm is clicked. The next button (9) opens the discover tables popup. Here the user is able to select an interface (wifi, ethernet) and scan for any connected tables. Only client id and secret needs to be entered manually. The next button opens a table filter pop up (10). Here the user is able to select if tables with missing credentials should be displayed or not. The next button (11) opens a page similar to main page in a new window. The purpose of this window is to display table statuses on a separate screen, while still having the ability to use the application. The last button (12) in the bottom bar is the settings button. It opens a popup where the user can switch language. There are currently two options to choose from, English and Norwegian. The settings also has a "delete all credentials" button that deletes all saved table credentials from disk.

Code-behind

The code-behind for the MainPage is shown in Listing 3. It shows that MainPage takes in a MainViewModel and a TablesContentView as constructor parameters. BindingContext is set to the MainViewModel. This enables data binding between view and viewmodel. From line 14, it shows an override of the OnAppearing() function. This function is called when the page becomes visible. In this case, it reloads the content of the page to prevent UI states from persisting unexpectedly. This might happen when the control caches previous states after the user navigates away from the page and returns.

Listing 3: MainPage code-behind

```
1  using KongsbergTableTracker.ViewModels;
2
3  namespace KongsbergTableTracker.Views;
4
5  public partial class MainPage : ContentPage
6  {
7      public MainPage(MainViewModel mainViewModel, TablesContentView
       tablesContentView)
8      {
9          InitializeComponent();
10         BindingContext = mainViewModel;
11         TablesContentView.Content = tablesContentView;
12     }
13
14     protected override void OnAppearing()
15     {
16         base.OnAppearing();
```

```

        if (BindingContext is MainViewModel mainViewModel)
        {
            var ReloadTablesContentView = new
            TablesContentView(mainViewModel);
            TablesContentView.Content = ReloadTablesContentView;
        }
    }
}
```

ContentViews

Why you used ContentViews ContentViews are used to make reusable controls that are defined in a seperate XAML file. [56] In the KTT application, ContentViews are used for the list of tables on MainPage and StatusPage, the InfoPanel on ActivePage and HistoryPage, the NavigationBar on most pages and TimeLine for a table. The InfoPanel is explained in more detail below.

The InfoPanel in Figure 21 is one of the ContentViews used in the KTT application. It contains several elements, and is used in ActivePage and HistoryPage. It is divided into four parts. Starting from the left are the TableStatus, JobStatus, ProductionInfo and RemainingTime. The first two elements are components that will be explained further in the next section. The ContentView is placed in the NameSpace "KongsbergTableTracker.Views.Components".

Listing 4: Include InfoPanel namespace in page

```
xmlns:local=
"clr-namespace:KongsbergTableTracker.Views.TableContextViews"
```

Listing 5: Implement InfoPanel on page

```
<ContentPresenter x:Name="InfoPanelViewHost" />
```

Listing 6: ActivePage constructor parameters

```
public ActivePage(TableContextService tableContextService,
    InfoPanelContentView infoPanelContentView)
```

Listing 7: Set content for InfoPanel

```
InfoPanelViewHost.Content = infoPanelContentView;
```

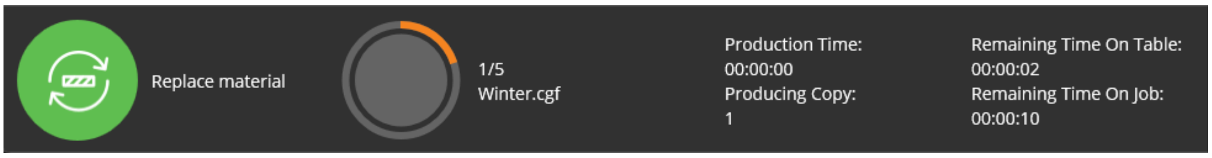

Replace material
1/5
Winter.cgf
Production Time:
00:00:00
Producing Copy:
1
Remaining Time On Table:
00:00:02
Remaining Time On Job:
00:00:10

Figure 21: InfoPanelContentView screenshot from ActivePage

Components

The ktt application also includes several different custom Components in the View folder. They are called HistoryTimeLine, ProgressWheel and StatusTimeWheel. The InfoPanelContentView (21) contains one of them, which is the ProgressWheel. The ProgressWheel is defined in a .cs file and consists of the ProgressWheel class (derived from GraphicsView) and a ProgressWheelDrawable (derived from IDrawable). The ProgressWheel works like a canvas and the ProgressWheelDrawable takes instructions and draw them.

Converters

The KTT application also utalize converters to transform data values between the ViewModel and the View through data bindings. One example used is the UtcTimeSpanToLocalConverter. TimeSpans from the Kongsberg HUB comes in UTC, so to create a better UX, a converter transforms the time from UTC to local time. Listing 8 shows the converter function.

Listing 8: UtcTimeSpanToLocalConverter

```
public object? Convert(object? value, Type targetType, object? parameter,
CultureInfo culture)
{
    if (value is TimeSpan utcTime)
    {
        var utcDateTime = DateTime.UtcNow.Date + utcTime;
        var localTime = utcDateTime.ToLocalTime().TimeOfDay;
        return localTime;
    }

    return value;
}
```

9.5 ViewModel

Table Context **TS |*OH***

In order to keep our table-specific viewmodels up to date with the received data, we have made a class to store both the table model, the viewmodels that target a single table and a table context messenger. Collectively, we have named these a TableContext, which is created and stored in the table context service whenever a table is introduced to the system, be it tables added manually by a user, tables loaded from disk or tables discovered on the network.

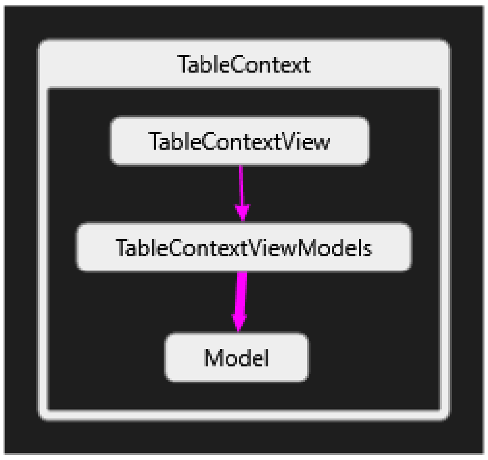


Figure 22: TableContext

Table Context Base **TS |*OH***

This is the base ViewModel that all the ViewModels inside the TableContext inherit from. It contains things that most or all of the TableContext viewmodels use, like references to the service provider, the TableContext/Meta messengers, the name of the table and a reference to the TableModel. The base ViewModel also receives the messages ResetViewModelMessage and LoadModelDataMessage. There are reset and load functions you can override in each child viewmodel.

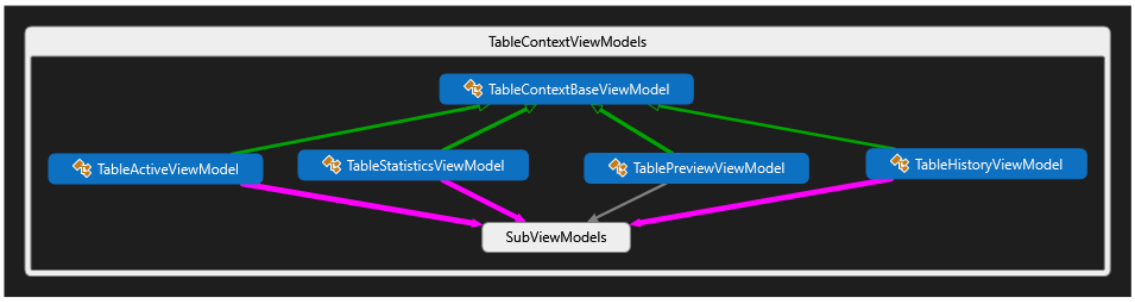


Figure 23: TableContextViewModels

Table Preview **TS |*OH***

The TablePreviewViewModel is responsible for providing the table info panels on the main page with information from their respective instance of the table model. This includes the table name, version of iPC, current job status, table image and connection status. By subscribing to the messages RunningJob, TablePropertyChangedMessage and ConnectionStatusChangedMessage, the TablePreviewViewModel can keep these fields updated with information from the model. Because ConnectionStatusChangedMessage can be sent from many places and those messages only converge in this viewmodel, these messages are propagated here from the per-table messenger to the meta messenger when a table connection changes status. This is done in order to update the filtered list of tables in TableContextService

Table Job Lists **OH |*TS***

The TableActiveViewModel is the binding context of the ActivePage. It takes the active jobs from the model and makes them accessible to the View. This is the same job list shown in iPC. This ViewModel implements job handling. It sorts the jobs based on actions from the view and updates the job list in real time by listening to the tablemodel. When pressing the add button in the view it executes the AddActiveJobAsync which opens a file explorer and lets you add a job in zip format to the table. You can also delete jobs by pressing the trash symbol in the view. The HistoryViewModel is the binding context for the HistoryPage, it is almost the same as ActiveViewModel except that it shows the history job list and removed adding job functionality.

Table Statistics OH |*TS*

One statistics entry is structured in a 3-layer structure. The first layer is the JobStatistics that shows all the information about the jobs produced. Then in the next layer is a list of JobRunStatistics, which is each time the job gets started and stopped. Then for each job run you have a list of JobLayersStatistics which holds the data for each layer. A layer has information about what type of tool is used. So, if multiple tools are used in one run you have a list of JobLayerStatistics inside the JobRunStatistic. That data is encapsulated in classes which are already loaded inside the model. Those models will be encapsulated into a viewmodel for extracting only the important data and to bind to the view. Those viewmodels we call SubViewModels. They are primarily used in the StatisticsViewModel but also represented as a JobTask which is used in Active and HistoryViewModel. They are mapped out in figure 24

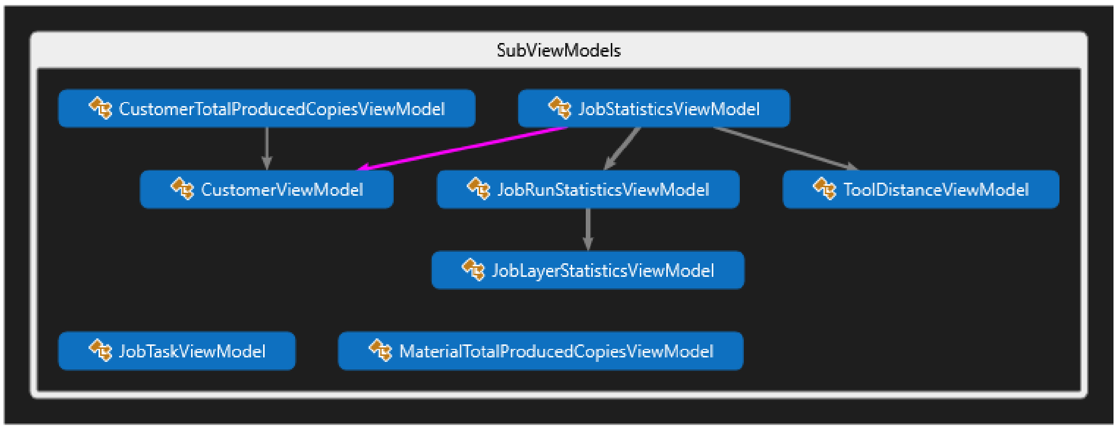

SubViewModels
CustomerTotalProducedCopiesViewModel
JobStatisticsViewModel
CustomerViewModel
JobRunStatisticsViewModel
ToolDistanceViewModel
JobLayerStatisticsViewModel
JobTaskViewModel
MaterialTotalProducedCopiesViewModel

Figure 24: SubViewModels

The statistics viewmodel is the binding context for the StatisticsPage. It gathers data from the statistics endpoint and generates information based on data important to the user. The StatisticsPage has 4 buttons you can press one daily, one weekly and a monthly which sets UTC time in the viewmodel and sends a request to the model about gathering information from that period. This then gets propagated back into the viewmodel with all the data in a list of JobStatisticsViewModels. With JobStatistics we can loop though the JobStatiscViewModel and gather the customers that each of the jobs was produced for and aggregate them using a dictionary with the CustomerViewModel as key and the amount of copies produced as value. If the job does not have a defined customer it just defaults to "not specified". Then you can go one step further and loop though the Runs. There you can do the same process only changing the key with a string representing the material name. Then the last layer is the production layer where the information about the distance traveled by each tool is stored. This can then be multiplied with the number of multipasses and number of copies to produce total distance cut in the material. This will generate 3 different dictionaries which represent production done by the table. They must then be cast into a display friendly viewmodel holding the customer and the number of copies, the material name and the amount of copies, and lastly the name of the tool and the distance cut. Those classes gets added into an ObservableCollection and displayed in the view.
It also sends a property trend message with the Status as key, gathering all the data about when the table was busy,idle or off. This is used to calculate the time the table was in each state. The endpoint returns a list of properties where each has the start and end point of the status. You can then loop though the list and search for each property and add the time from that instance to the next shift in status. This is then calculated into total busy, idle and off times inside the timespan. But the first instance of status inside the timespan might not be defined with a property trend you have to fill that in with unknown time. From all of that you can find the utilization score of the table.

External **OH |*TS***

With the implementation of the table specific viewmodels that hold information about a single table there are also external viewmodels connected to the display and handling of multiple tables. The external viewmodels are Main, Timeline and Meta Statistics. There are also popups connected to buttons inside each of the external viewmodels. Some of the popups have their own viewmodel and some do not, depending on the required functionality of the page. The external viewmodels are shown in Figure: 25

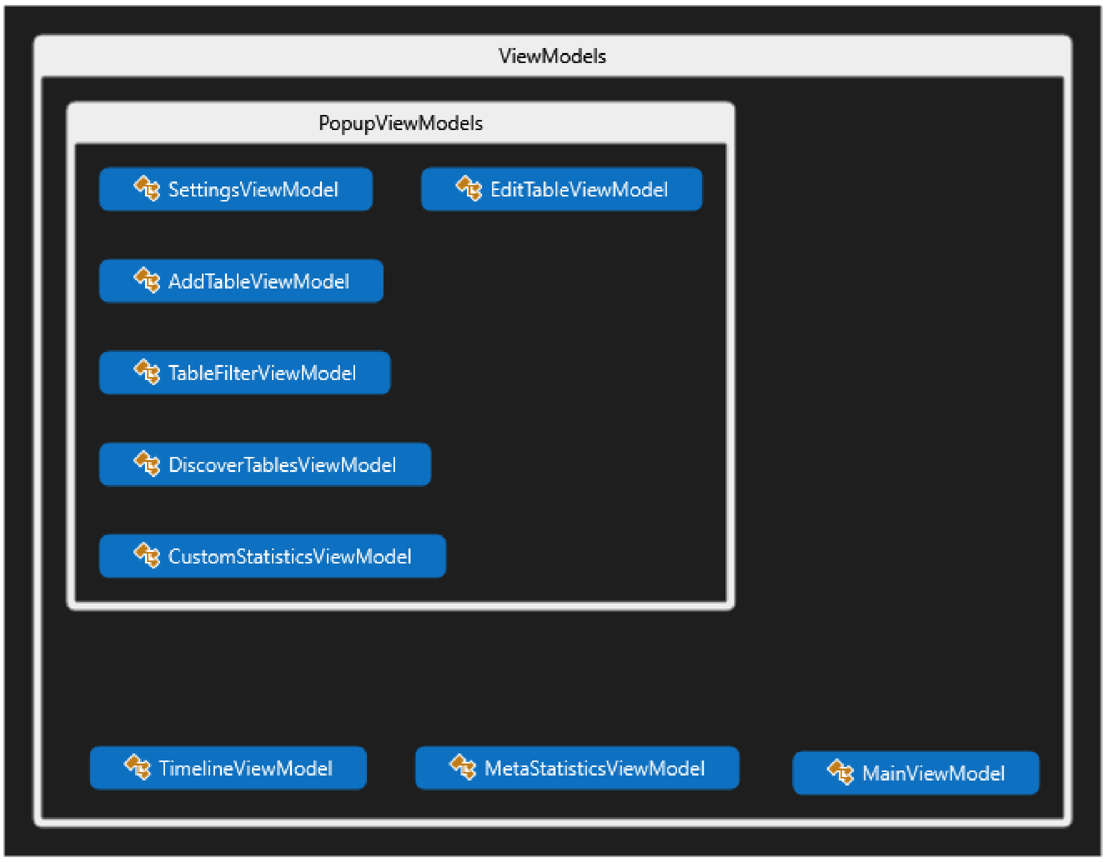


Figure 25: External ViewModels

Main OH |*TS*

The first page you enter is the MainPage. This page has a corresponding viewmodel called the MainViewModel. It is used for data binding as we will describe in section 9.7. The main viewmodel constructor injects the IServiceProvider container from .NET MAUI and the TableContextService. The MainViewModel is responsible for getting the TableContext list from the TableContextService for display. It also handles the TableContexts. The MainViewModel has many commands the view can bind to. They all correspond to a way of handling the TableContext list or navigation. The GoToTable command takes a TableContext as a parameter. The command sets the parameter as the ActiveTable inside the TableContextService. Then navigates to the ActivePage which uses the ActiveTable's ActiveJobsViewModel member to set the binding context. The AddTable command launches a popup for adding a TableContext. The EditTable command launches a popup almost the same as AddTable. The popup makes you able to edit the credentials on that table. The DeleteTable command removes a TableContext from the TableContextService. The ReconnectTable command takes the TableContext as a parameter and calls the function InitializeTable on the parameter. Then you have commands for navigation to the DiscoverTablePopup. The last commands are used to navigate to the TableFilterPopup and SettingsPopup. All the popups are described in section 9.5

TimeLine TS |*OH*

This viewmodel has a Task that is activated when the TimelinePage is opened. It simply runs a loop that waits 5 seconds before it sends an UpdateTimestampsMessage over the MetaMessenger. This message then reaches each instance of TimelineContentView to ensure the timestamps have an opportunity to update regularly without each and every one of them having their own Task with delay.

Meta Statistics OH |*TS*

The TableStatisticsViewModel holds all the data for a specific table. Then there is the external viewmodel format of the StatisticsPage. The MetaStatisticsViewModel holds commands to buttons that request data from a timespan defined by the user. It sends a request to the Table StatisticsViewModel that gathers the same data as it did standalone. It knows that it was requested by the MetaStatisticsViewModel and returns itself. The MetaStatiticsViewModel aggregates the same information using dictionaries. This is then displayed as an ObservableCollection of customers, material and tool distances. It also displays the status time and the utilization by adding up the times.

Popups TS |*OH*

AddTable

This viewmodel handles the verification of user input when adding a table manually through the Add Tables popup window. If the user clicks confirm and the entry fields all pass the verification, then a TableContext is created with the information provided by the user and added to the list in TableContextService. This viewmodel uses a TableEntryFieldValidator that checks if each field contains a value that can be used. The confirm button is disabled when a field has an invalid value

CustomStatistics
This viewmodel handles custom timespans provided by the user in the form of a before date and an after date from the popups in TableStatisticsPage and MetaStatisticsPage

DiscoverTables
This viewmodel handles the interactions between the user and the TableScan service. The user can select a network interface and request a scan on it. The interface list is filtered to exclude network interfaces that are not connected to class C networks. An abort scan button is shown while a scan is in progress. When a scan is completed, the queue of tables returned from the TableScan service is added to the TableContext service

EditTable
This viewmodel is very similiar to the AddTable viewmodel. The major difference is that it doesn't add a new table, but edits an existing one and re-initializes it when the user clicks confirm

Settings
This viewmodel handles user input from the Settings popup. Currently, the only settings are changing language and deleting all credentials from both memory and disk.

TableFilter
This viewmodel sets table filters based on user input. The filters are passed to the TableContext service. Currently, the only filter available is to filter out tables that are not connected to their respective Kongsberg HUB

9.6 Services **TS** | ***OH***

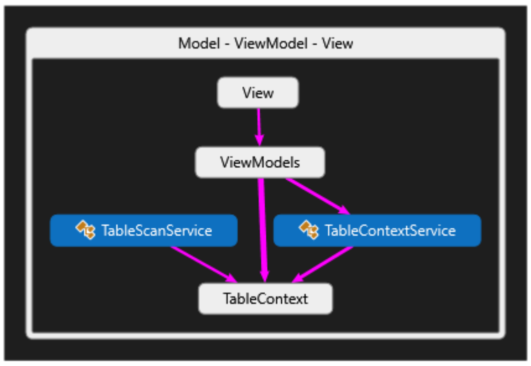


Figure 26: TableContext-ViewModel-View

Table Context Service

The TableContext service encapsulates a list of every TableContext that has been created. When a table is added/removed to/from the list of TableContexts, the name and IP address of every table in the table list is serialized to JSON and stored on disk. When the application is launched, it will load the list from disk (if it exists) and put the contents back into the list of TableContexts. The TableContext service also has a filtered list of TableContexts, where user defined filters determine which TableContexts go from the main list of TableContexts into the filtered list. If the user has not enabled any filters, the filtered list is identical to the main list. By the time this was written, there was only one filtering option implemented. The option to filter out tables that did not have a connection to a Kongsberg HUB. However, the ground work for adding filtering options has been laid out, so adding more filters later should be fairly trivial. In order to keep the filtered list up to date, the TableContext service has subscribed to Connection-StatusChangedMessage through the meta messenger, so that the function to update the filtered list is called whenever a table changes connection status.

Table Scan Service

Having to manually input a bunch of table names and Internet Protocol (IP) addresses is annoying, so we added a function to scan the network for cutting tables. This is done by sending a request to every single IP address on the network using one of the Kongsberg HUB endpoints that don't require authentication to get a response and checking the response. The user selects a network interface to scan on, and a list of IP addresses is generated based on the IP address and subnet mask associated with the user selected network interface. To generate this list, we first get the network and broadcast addresses. To get the network address, do a bitwise AND operation between the host IP address and the subnet mask. As for the broadcast address, do a bitwise OR operation between the host IP address (can optionally be substituted for the network address if it is already calculated) and the bitwise inverse of the subnet mask. Both the network address and subnet mask are converted to unsigned integers. By subtracting the network address from the broadcast address, we get the amount of IP addresses on the network. Then we can run a loop and add every IP address from the network address up to the broadcast address (not including either as they are not eligible for assignment to any device) to a list. From there we run a Task for every IP address in the list. Each Task sends a request to the IP address and checks if the response is successful, in which case it tries to do a reverse DNS lookup on the IP address just to have a name for the table. In case the reverse DNS fails, a default name of ”Unknown” is set for the new table. At the end of the Task, when the table has been discovered and the reverse DNS request is completed, the new table is added to a queue. After all the Task objects have either finished or expired, the list is returned to the DiscoverViewModel, where the list of discovered tables is added to the list of all tables in the TableContext service.

9.7 Data Flow

Messenger **OH |*TS***

The main messaging system of the application is the community toolkit IMessenger. That was chosen after a technical design document was created mapping out the pros and cons of IMessenger, Events and Direct push. The document is located in Appendix I. The IMessenger is a messenger thats sends classes. Each of those classes is defined in the different layers of the system. There are currently 2 messengers in the application. One is named TableContextMessenger and is injected into every TableContextViewModel. That handles the messages between the TableModel and the TableContextViewModels. The TableModel sends out a message for Reset, Init and each of the events. They are received by a viewmodel that has instantiated the IRecipient and the viewmodel makes the function public void receive(typeof(Message)). Then each time the message is sent, the viewmodels can do the appropriate data updating. Then you have the MetaMessenger, it is used to communicate between the external viewmodels and the TableContextViewModels. The implementation is the same as the TableContextMessenger and is currently used for the MetaStatisticsViewModel to send request for data and the ConnectionStatusChanged message.

Binding mechanics **OH |*TS***

As mentioned earlier, each of the views has their own viewmodel. The viewmodel is injected into the code-behind of the views, then assigned as the binding context for that page. Each of the viewmodels inherits from ObservableObject class. The properties are bound to the view using {Binding ....} command inside the XAML page. All the properties inside the view are prefixed with [ObservableProperty] which is a function added by the NuGet package named CommunityToolKit .Mvvm. The properties are then registered to the INotifyPropertyChanged automatically. This makes updates in the view automatic. The commands implemented in the viewmodel are either using ICommand or the Toolkit version [RelayCommand], those are then bound to the view.

9.8 Test Results **MO |*EP***

In this chapter, we will present the results from the different tests we performed during the project. This includes unit testing, integration testing, system testing, acceptance testing. We describe what was tested, what worked well, and which bugs or problems we discovered. The goal is to show how the system behaved during testing and what we learned from it.

Unit Testing Results

Unit Test Example: GetActiveJobsTest.cs

This unit test checks the method `GetActiveJobs` from the `ApiServiceWithClient` class. The purpose of the method is to call a REST API to fetch active job data and convert the response into usable objects for the system.
The test was written using the xUnit framework and follows the Arrange–Act–Assert pattern:

- **Arrange:** A fake HTTP handler is set up to simulate different API responses. This allows us to control the input to the method without needing a real server.
- **Act:** The `GetActiveJobs` method is called using the fake client.
- **Assert:** The returned result is compared to the expected outcome to check that the method behaves correctly.

Test Scenarios:

1. **Scenario 1: API returns valid job data**
   - A JSON string with one job is returned from the fake API.
   - The test checks that the result is not null.
   - It also checks that the returned list contains one job with the correct ID and name.
2. **Scenario 2: API returns `"null"`**
   - A plain JSON value `"null"` is returned.
   - The method is expected to return null and not throw any exceptions.

Result: Both test scenarios passed. The method returned the correct results and handled the edge case without crashing.
Lessons learned:

- Using custom HTTP handlers is an effective way to simulate API behavior in unit tests.
- Testing with edge cases, like null responses, makes the system more robust.
- Small, focused unit tests help make the code easier to maintain and understand.

For the full unit testing report, see Appendix G.

Integration Testing Results

Integration testing was performed manually. The team used the application and verified that the frontend correctly communicated with the backend. Views were checked, buttons were pressed, and data was confirmed to be correct. This helped discover bugs that were not covered by unit tests. Now the application runs smooth and nothing unexpected happens when you navigate through the application using the different buttons.

System Testing Results

During this testing phase, we found some bugs. One problem happened when we moved too fast between pages. If we clicked quickly or changed screens before the app was ready, it sometimes gave an error or crashed. We also had some issues with the timeline. Sometimes the timeline did not show the correct data. These bugs were fixed by improving how the app loads and updates the data. System testing helped us find these problems that we did not see in unit or integration testing.

Acceptance Testing

At the end of the project, the team had a final meeting with the stakeholder from KPCS. In this meeting, the full application was demonstrated. The group walked through all the main features, including table overview, job handling, and the status display. The stakeholder followed the walkthrough and confirmed that the system worked as expected. All the agreed requirements were met, and the stakeholder was satisfied with the result. This meeting marked the successful completion of the acceptance testing.

Bugs discovered

During testing, we found some bugs in the system. One bug happened when we moved too quickly between pages in the app. If we clicked fast or switched views quickly, the app sometimes gave an error or crashed. This did not happen every time, but often enough that we noticed it.

We also had some problems with the timeline. Sometimes the timeline did not show the correct data, or it was not updated when it should have been. This was likely because the app did not get the new data in time, or because things were happening at the same time in the background. We worked on fixing this by improving how the app loads and updates the timeline.

Summary

We tested the system in different ways to make sure everything worked as it should. Unit tests helped us check small parts of the code and making sure the functions behaved like they should. These tests made it easier to find bugs early and made the code more stable. Integration testing helped us check that the frontend and backend worked well together. System testing showed us how the app worked when everything was running, and helped us find bugs like crashes when clicking too fast or problems with the timeline. In the acceptance test, the stakeholder saw the full system and confirmed that it met the requirements.
The tests showed that most of the system worked well, but also helped us find and fix some problems. After fixing the bugs, the system became more stable and ready to use.

10 Results MO |*EP*

Regression Test

The test plan was executed together with two employees from KPCS. We went through all the planned tests step by step to ensure that the core functionality of the system still worked as expected. This included checking things like table setup, job handling, dashboard features, and the timeline view. The KPCS employees observed the testing and gave feedback along the way. Below are the results from the regression test: for the full regression test report see Appendix H

Table 8: Regression Test Example – Establish Connection and Add Tables

Test ID	T-1
Test Name	Establish connection and add tables
Requirements Tested	R-1.1.1, R-1.1.2, R-3.3.3, R-5.1.1
Steps	1. Open Kongsberg Hub Client Configurator. 2. Press "Add Table" and enter IP and Hostname. 3. Enter Name, IP address, ClientID, and ClientSecret. 4. Press Confirm. 5. Repeat steps 1–4 to add a second table.
Expected Result	The table appears in the GUI with the correct name, connection status, image, and job data. After repeating the steps, two tables should be visible.
Tester	–
Date	–
Pass/Fail	–

11 Product Risk Analysis MO |*EP*

While the project team focused on managing risks during the development phase, it is also important to consider risks related to the product itself. These risks affect the final user experience, the system's long-term usability, and the reliability of the delivered software. Product risks include potential problems such as incorrect functionality, system instability, or unclear user interfaces. These issues can lead to user dissatisfaction or increased maintenance in the future. Below are some identified product-related risks that were considered during the project
As explained earlier in Section 5.3, the risk evaluation is based on the RPN method. This helps compare different risks by using probability, impact, and detection values.

The following table presents the most important product-related risks identified in the project.

Product Risk	Explanation	Probability (1–5)	Impact (1–5)	Detection (1–5)	RPN	Overall Risk
Software bugs	The software may crash or behave incorrectly in certain cases.	3	3	2	18	MODERATE
Incorrect data display	Job or table data might not show the correct status or timing.	1	4	3	12	LOW
User misunderstanding	The user interface could be unclear or confusing, causing wrong usage.	2	2	2	8	LOW
Integration with backend fails	The connection to the Kongsberg HUB might be unstable or slow.	1	3	3	9	LOW
Missing functionality	Some features expected by the customer might not be implemented fully.	2	3	3	18	MODERATE
Long term usage	The system might not perform well over time if more tables or jobs are added in the future.	3	4	2	24	MODERATE
High memory usage during data processing	The system may use too much memory when processing or retrieving data over specific time periods, which could lead to performance issues or crashes.	3	3	2	18	MODERATE

Table 9: Product risk table

12 Challenges

12.1 Project **MO |*EP***

During the project, we had some challenges that affected our work. One group member, Elvin, was in Belgium as an exchange student. His semester there overlapped with ours, so he had to try and contribute to the project while also preparing for his exams before coming back to Norway. This made communication and planning a bit more difficult in the early phase.

In the beginning, most of us also had little to no experience with C# but we had written a lot of c++ code so it was not so different from that. This meant we had to spend extra time learning the language and tools.
We had problems with the working space too. Our usual room in Bergseminaret, was under renovation and had a lot of noise. We then got a new room which had seen better days. Two of the group members wanted to return to our old room despite the noise. So we settled it like real men would, and that is of course with rock paper scissors.

Even though we had some small challenges during the project, we worked well together as a group. We had good communication and helped each other when something was difficult. We did not have any major conflicts or problems in the group, and we are happy with how we worked as a team throughout the project.

12.2 Technical

No access to iPC from the start **TS |*OH***

At the start of the project, we did not have license keys to iPC and were only able to test our application when we were at KPCS. We initially tried to solve this by making a fake Kongsberg HUB in Python using the Flask framework to deliver dummy data on the same endpoints as the Kongsberg HUB. The first few endpoints from the Kongsberg HUB API were simple enough to add to the Python script, but it would have taken way too long to make a complete fake Kongsberg HUB. Thankfully, we received license keys from KPCS, so we could scrap the Python script and free up those resources to work on more important things.

Development versions of the HUB **TS |*OH***

Throughout the project, we have been testing our application against different development builds of the Kongsberg HUB. Some of those versions do not always behave as expected. Sometimes the Kongsberg HUB did not include certain key data in the event messages it sent to our application, which made our application not respond in the way we had anticipated. This was confusing at first, and we had to investigate the cause of the problem through extensive debugging messages. It wasn't until we inspected the data received from the Kongsberg HUB in Postman that it became clear that we had not done anything wrong on our end. In situations like this, where the cause of the error is outside our area of responsibility, we deduced that the appropriate course of action is to send a bug report to KPCS and leave our application as is.
Kongsberg HUB also doesn't expose enough data for us to be able to accurately replicate

the job/copy counter (the orange wheel). There will be some discrepancies between the job/copy counter in our app and the one in iPC.

JSON deserialization TS |*OH*

The live data we receive from the Kongsberg HUB is sometimes in the format of an object (or nested objects) where one or more members are set to null. Our original code could not handle updates like this and would overwrite the entire object when only one of its fields needed an update. We were unable to solve this problem ourselves and used generative AI to come up with a solution. See TableModel.UpdateJobTask(JobTask, JobTask) and its helper function UpdateJobTaskMemberOfClassType(object?, object, Type)

Scan service on bigger networks TS |*OH*

The scan service uses a lot of system resources in its current state when trying to run a scan on a network bigger than a class C network. As an example, RAM usage has been observed well above 2GB. For this reason, a seemingly arbitrary restriction was placed on the size of networks that can have scans performed on them. This restriction can easily be removed at some point in the future when the scan service is rewritten to be more resource efficient.

Auto-generated files EP |*MO*

With a .NET MAUI project, there are many files that are generated automatically. It can be challenging since all group members are new to the framework. Issues can occur when two different branches are merged, and the auto-generated files have been modified by the helpers.

Cross-platform EP |*MO*

The initial plan was to develop an application that would work on both Windows and Android. It was quickly decided that the team lacked the necessary ”man-power” to deliver a good result on both platforms.

13 Conclusion EP |*MO*

The goal for this project was to create a prototype application that could be used for remote monitoring of multiple tables on a Local Area Network (LAN). The application should display information from the tables in real time, and also include features such as table status, job list, job status, adding and deleting jobs. This application would be a helpful tool for operators, managers, and developers alike. During the development process, all members had to learn new coding languages, libraries and architectures. Even with limited knowledge, the team systematically worked towards a solution, one step at a time. The challenges that occured during the process, was solved with good communication, both within the team, but also by consulting supervisors. By using tools such as Draw.io, Jira, and Git, the team was able to work systematically on pre-assigned tasks. Progress was smooth, and the skill level of the team increased visibly every week. The outcome of the project was a success. The prototype application was completed within the time frame, and feedback from supervisors was positive. The application satisfied almost all the requirements set, with the exception of cross-platform. The teamwork within the group has been above all expectations, and all members are proud of the final product, Kongsberg Table Tracker.

14 Future Work

Since this project is to be handed over to KPCS at the end of the semester, it is important to include information that will be helpful to the team that takes over the project. That is the purpose of this chapter.

14.1 Cross platform EP |*MO*

One of our initial requirements was to include support for cross-platform in the KTT application. After consulting with kpcs early in the project, we came to an agreement to only focus on Windows development, as it is the most important target platform. The reason this issue was brought up, was the realization of how time consuming it would be to focus on multiple platforms at the same time. With only four members on the group, it was decided that we would drop the requirement of cross-platform. It is however important to note that the foundation is already there. The application does launch in the android emulator, but is missing the touch-friendly controls to be usable.
Another thing that should be implemented is the use of more ContentViews. The application has pages that are very similar, StatisticsPage and MetaStatisticsPage, MainPage and StatusPage, and ActivePage and HistoryPage. Only the MainPage and StatusPage share a ContentView. It would be a good idea to do the same for the other pages, in order to reduce duplicated code.
It has also been pointed out by an external tester that it would be helpful to include tooltip of status text on the table preview.
Another UI related improvement is adding highlighting to the navigation bar, so the user can better see which page they are currently on.

14.2 Server certificates TS |*OH*

When we began using HttpClient for communicating with the Kongsberg HUB, we didn't handle server certificate validation properly. It was bypassed altogether, leaving a security hole in our application. This must be remedied before public release.

14.3 Scan service efficiency TS |*OH*

Make the TableScan service more resource efficient so that the class C network restriction can be removed.

14.4 Better handling of table editing TS |*OH*

Currently, in the edit table popup, editing any field (or even no fields at all, just clicking confirm) will force the table to re-initialize. Some checks on the new values should be done to avoid potentially unnecessary refreshing of data.

14.5 Preferences TS |*OH*

Insufficient research was conducted prior to implementing user preferences, resulting in a subpar homemade solution when a perfectly good easy-to-use option already exists in .NET, simply called Preferences. The homemade solution should be replaced with Preferences.

14.6 Installer **TS |*OH***

An installer should be made, to allow users to install the application and its prerequisities. We made an attempt at this, but it refused to install on any computer due to a lack of a trusted certificate. No further work was put into this due to other tasks taking priority.

15 References

[1] K. P. C. Systems, “Kongsberg ultimate 64,” 2025, accessed: 10-Feb-2025. [Online]. Available: https://www.kongsbergsystems.com/en/cutting-systems/tables/ultimate/ultimate-64

[2] ——, “i-cut production console,” 2025, accessed: 15-May-2025. [Online]. Available: https://www.kongsbergsystems.com/en/i-cut-production-console

[3] davidbritch, “Layouts,” 2024, accessed May 17, 2025. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/maui/user-interface/layouts/?view=net-maui-8.0

[4] K. P. C. Systems, “Digital cutters for every application,” 2025, accessed: 15-May-2025. [Online]. Available: https://www.kongsbergsystems.com/en/

[5] ——, “Advanced digital cutting machines for signage, display & corrugated production,” 2025, accessed: 15-May-2025. [Online]. Available: https://www.kongsbergsystems.com/en/solutions/digital-cutting-machine

[6] Thorpe. (2025) What is the digital cutting machine? Accessed 18-May-2025. [Online]. Available: https://www.igolden-cnc.com/what-is-the-digital-cutting-machine/

[7] Wikipedia, “Tooling,” 2025, accessed: 15-May-2025. [Online]. Available: https://en.wikipedia.org/wiki/Milling_(machining)

[8] K. P. C. Systems, “Tooling,” 2025, accessed: 15-May-2025. [Online]. Available: https://www.kongsbergsystems.com/en/cutting-systems/tooling

[9] ——, “Kongsberg c series: Technical specifications,” 2025, accessed: 15-May-2025. [Online]. Available: https://www.kongsbergsystems.com/en/kongsberg-c

[10] ——, “i-cut production console,” 2025, accessed: 15-May-2025. [Online]. Available: https://www.kongsbergsystems.com/en/i-cut-production-console

[11] M. Kvalbein, “Personal communication with martin kvalbein, software engineer at kpcs,” 2025, conducted on 16 May 2025.

[12] K. P. C. Systems, “About the company,” 2025, accessed: 10-Feb-2025. [Online]. Available: https://www.kongsbergsystems.com/en/about/company

[13] ——. (2023) About. Accessed 18-Feb-2025. [Online]. Available: https://www.linkedin.com/company/kongsbergpcs/about/

[14] davidbritch, “What is .net maui?” 2025, accessed: 15-May-2025. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-8.0

[15] Wikipedia, “.net framework,” 2025, accessed: 15-May-2025. [Online]. Available: https://en.wikipedia.org/wiki/.NET_Framework

[16] davidbritch, “Xaml hot reload for .net maui,” 2025, accessed: 15-May-2025. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/maui/xaml/hot-reload?view=net-maui-8.0

[17] ——, “Xaml,” 2025, accessed: 15-May-2025. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/maui/xaml/?view=net-maui-8.0

[18] Wikipedia, “Markup language,” 2025, accessed: 16-May-2025. [Online]. Available: https://en.wikipedia.org/wiki/Markup_language

[19] C. Hashemi-Pour, “user interface (ui),” 2025, accessed: 17-May-2025. [Online]. Available: https://www.techtarget.com/searchapparchitecture/definition/user-interface-UI

[20] S. Levy, “graphical user interface,” 2025, accessed: 17-May-2025. [Online]. Available: https://www.britannica.com/technology/graphical-user-interface

[21] Microsoft. (2025) Visual Studio: IDE and Code Editor for Software Developers and Teams. Accessed 11-Feb-2025. [Online]. Available: https://visualstudio.microsoft.com/

[22] JetBrains, “Jetbrains: Developer tools for professionals and teams,” 2025, accessed: 17-May-2025. [Online]. Available: https://www.jetbrains.com/lp/rider-maui

[23] davidbritch, “Get started with .net maui xaml,” 2025, accessed: 15-May-2025. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/maui/xaml/fundamentals/get-started?view=net-maui-8.0

[24] m. M. davidbritch, Saccomani, “Visual states,” 2024, accessed May 17, 2025. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/maui/user-interface/visual-states?view=net-maui-8.0

[25] T. bijington, jfversluis, “Popup,” 2024, accessed May 17, 2025. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/communitytoolkit/maui/views/popup

[26] g. davidbritch, “Layouts,” 2024, accessed April 16, 2025. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/maui/user-interface/layouts/?view=net-maui-8.0

[27] j. davidbritch, marius-bughiu, “Controls,” 2024, accessed May 17, 2025. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/maui/user-interface/controls/?view=net-maui-8.0

[28] Microsoft, “Model-view-viewmodel (mvvm) pattern,” 2024, accessed March 25, 2025. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm

[29] (2025) Introducing json. Accessed 18-May-2025. [Online]. Available: https://www.json.org/json-en.html

[30] Microsoft, “System.text.json namespace,” 2025, accessed 18-May-2025. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/api/system.text.json

[31] ——, “How to write .net objects as json (serialize),” 2025, accessed 19-May-2025. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json/how-to

[32] Cisco. (2025) Configure ip addresses and unique subnets for new users. Accessed 18-May-2025. [Online]. Available: https://www.cisco.com/c/en/us/support/docs/ip/routing-information-protocol-rip/13788-3.html

[33] Cloudflare. (2025) What is dns? — how dns works. Accessed 19-May-2025. [Online]. Available: https://www.cloudflare.com/learning/dns/what-is-dns/

[34] ——. (2025) What is reverse dns? Accessed 19-May-2025. [Online]. Available: https://www.cloudflare.com/learning/dns/glossary/reverse-dns/

[35] Mozilla. (2025) An overview of http. Accessed 18-May-2025. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Overview#http_messages

[36] Microsoft, "Make http requests with the httpclient class," 2025, accessed 15-May-2025. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/fundamentals/networking/http/httpclient

[37] Wikipedia, "Server-sent events," 2025, accessed 18-May-2025. [Online]. Available: https://en.wikipedia.org/wiki/Server-sent_events

[38] Microsoft, "Httpclient.getstreamasync method," 2025, accessed 18-May-2025. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpclient.getstreamasync?view=net-9.0

[39] Wikipedia, "Rest," 2025, accessed 18-May-2025. [Online]. Available: https://en.wikipedia.org/wiki/REST

[40] Nickolay Bakharev. (2023) Unit testing: Definition, examples, and critical best practices. Accessed 8-Apr-2025. [Online]. Available: https://brightsec.com/blog/unit-testing/

[41] GeeksforGeeks, "Integration testing – software engineering," 2025, accessed: 2025-05-18. [Online]. Available: https://www.geeksforgeeks.org/software-engineering-integration-testing/

[42] Randall W. Rice. (2022) What is system testing? an in-depth guide. Accessed 8-Apr-2025. [Online]. Available: https:https://www.practitest.com/resource-center/article/what-is-system-testing-form/

[43] Guru99. (2024) System testing in software testing. Accessed 18 May 2025. [Online]. Available: https://www.guru99.com/system-testing.html

[44] S. Das. (2024) What is acceptance testing? Accessed 8-Apr-2025. [Online]. Available: https://www.browserstack.com/guide/acceptance-testing

[45] Guru99. (2024) What is regression testing? learn with example. Accessed May 2025. [Online]. Available: https://www.guru99.com/regression-testing.html

[46] SixSigma.us, "Risk priority number (rpn) in fmea," 2024, accessed: 17 May 2025. [Online]. Available: https://www.6sigma.us/six-sigma-articles/risk-priority-number-rpn/

[47] K. Schwaber and J. Sutherland, "The scrum guide," 2020, accessed: 10-Feb-2025. [Online]. Available: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf

[48] Microsoft. (2025) .net multi-platform app ui (.net maui). Accessed 11-Feb-2025. [Online]. Available: https://dotnet.microsoft.com/en-us/apps/maui

[49] Overleaf. (2025) Latex editor features & benefits. Accessed 11-Feb-2025. [Online]. Available: https://www.overleaf.com/about/features-overview

[50] JGraph. (2023) draw.io. Accessed 11-Feb-2025. [Online]. Available: https://www.drawio.com/

[51] JetBrains. (2025) Webstorm: The javascript and typescript ide, by jetbrains. Accessed 11-Feb-2025. [Online]. Available: https://www.jetbrains.com/webstorm/

[52] Microsoft. (2023) Framework design guidelines. Accessed 20-May-2025. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/standard/design-guidelines/

[53] BrowserStack, "Introduction to code based testing and its importance," 2025, accessed: 17 May 2025. [Online]. Available: https://www.browserstack.com/guide/code-based-testing

[54] Microsoft, "Unit test basics," 2025, accessed: 19-May-2025. [Online]. Available: https://learn.microsoft.com/en-us/visualstudio/test/unit-test-basics?view=vs-2022

[55] F. Awsome. (2025) Free license — font awesome. Accessed 19-May-2025. [Online]. Available: https://fontawesome.com/license/free

[56] g. davidbritch, "Contentview," 2024, accessed May 20, 2025. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/maui/user-interface/controls/contentview?view=net-maui-8.0

[57] Microsoft. (2024) Messenger. Accessed 20-May-2025. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/communitytoolkit/mvvm/messenger

[58] ——. (2025) Handle and raise events. Accessed 20-May-2025. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/standard/events/

A User Stories

ID	**US-1**
Title:	Status Change
User story:	**As an** operator
	I want to see when a table on the local network has changed status or needs interaction
	So that the factory can maintain optimal efficiency, and I can focus on other tasks while the table is running
Acceptance Criteria:	**Given** I have the application open and it is connected to the table.
	When the table status changes or requires interaction
	Then I should be notified

ID	**US-2**
Title:	Job Preparation
User story:	**As a** floor manager
	I want to manage jobs remotely
	So that I can stay productive in my office
Acceptance Criteria:	**Given** I have the application open, and it is connected to the table
	When I upload and delete jobs
	Then the table receives the change and displays it

ID	**US-3**
Title:	Data view
User story:	**As an** operator
	I want to observe all the relevant job data on my table
	So that I can keep track of my table without being physically present.
Acceptance Criteria:	**Given** I have connected the table to the application
	When I open the application
	Then I can view relevant data in a structured way.

ID	**US-4**
Title:	Custom view
User story:	**As an** operator
	I want a system that I can customize to view relevant data
	So that I dont get flooded with irrelevant information
Acceptance Criteria:	**Given** I have the application running
	When I move and filter data
	Then the application shows the filtered data.

ID	**US-5**
Title:	Multiple tables
User story:	**As a** shop-floor manager
	I want to have a visual interface to present data from multiple tables
	So that I can keep track of job data from all my tables in one place
Acceptance Criteria:	**Given** I have mulitple tables connected to the same local network
	When I open the application and add the table credentials
	Then I can see an overview of the tables

ID	**US-6**
Title:	Job statistics report
User story:	**As a** shop-floor manager
	I want to see daily, weekly and monthly production statistics
	So that I have a full overview of my production
Acceptance Criteria:	**Given** the table has produced jobs in the past
	When user requests a report
	Then a report is generated based on the data from the table

ID	**US-7**
Title:	Cross-platform compatibility
User story:	**As a** shop-floor manager
	I want to see table information on diffrent types of devices
	So that I can stay updated wherever I am
Acceptance Criteria:	**Given** I have the application open
	When using either a desktop or a handheld device
	Then it should display the corresponding user interface

B Use case

ID	UC-1.1
Use case:	Table status change
Description:	The system shows the operator when a table on the local network changes status.
Actor:	Operator

ID	UC-1.2
Use case:	Notification handling
Description:	The system shows information about the status of the table
Actor:	Operator

ID	UC-2.1
Use case:	Add job remotely
Description:	The systems allows the operator to add jobs from the table using a separate device
Actor:	Shop-floor Manager

ID	UC-2.2
Use case:	Remove job remotely
Description:	The systems allows the operator to remove jobs from the table using a seperate device
Actor:	Shop-floor Manager

ID	UC-2.3
Use case:	Remote job configuration
Description:	The systems allows the operator to configure job setttings remotely before starting the job
Actor:	Shop-floor Manager

ID	UC-3.1
Use case:	View job data remotely
Description:	The system allows the operator to view the relevant job data from a remote device
Actor:	Operator

ID	UC-3.2
Use case:	Data synchronization
Description:	The system ensures that the job data is updated in real-time
Actor:	Operator

ID	UC-3.3
Use case:	Data presentation
Description:	The system organizes and displays the job data in a structured way
Actor:	Operator

ID	**UC-4.1**
Use case:	Data customization
Description:	The applications graphical user interface is customizable to display desired data
Actor:	Operator

ID	**UC-5.1**
Use case:	Multiple table support
Description:	The application shows an overview of the different tables and their current status
Actor:	Shop-floor Manager

ID	**UC-5.2**
Use case:	Scan for multiple tables
Description:	The application can scan for tables on the LAN
Actor:	Shop-floor Manager

ID	**UC-6.1**
Use case:	Generate production statisticts
Description:	The system generates a production report based on daily, weekly and monthly statistics.
Actor:	Shop-floor Manager

ID	**UC-6.2**
Use case:	Display Production Report
Description:	The system presents the production report in a stuctured format when requested by user.
Actor:	Shop-floor Manager

ID	**UC-7.1**
Use case:	Handheld device
Description:	The system work on both a desktop and a handheld device
Actor:	Shop-floor Manger

C System requirements

ID	User story	ID	Use case	ID	Requirement	Priority
US-1	Status Change	UC-1.1	Table status change	R-1.1.1	The application shall establish connection to the cutting table	A
				R-1.1.2	The application shall retrieve table status from the HUB	A
		UC-1.2	Notification handling	R-1.2.1	The application shall display the status change in exactly the same way as the table	A
				R-1.2.2	The application shall show what the operator needs to fix	B
US-2	Job Preparation	UC-2.1	Add job remotely	R-2.1.1	The application shall allow the operator to add a zip file to the table	B
		UC-2.2	Remove job remotely	R-2.2.1	The application shall allow the operator to remove jobs from the table	B
		UC-2.3	Remote job configuration	R-2.3.1	The application shall allow the operator to set the copies for the job	C
US-3	Data view	UC-3.1	View data remotely	R-3.1.1	The application shall display job data from a table elsewhere on the network	A
		UC-3.2	Data synchronization	R-3.2.1	The application shall handle events sent from the HUB	A
				R-3.2.2	The application shall query the HUB for potentially outdated information on startup	A
		UC-3.3	Data presentation	R-3.3.1	The application shall have a list of jobs	A
				R-3.3.2	The application shall display the job data when the job is clicked	B
				R-3.3.3	The application shall display an overview of tables	A
				R-3.3.4	The application shall provide a visually appealing and well-structured GUI	B
US-4	Custom view	UC-4.2	Data customization	R-4.1.1	The application shall let you reorganize items	C
				R-4.1.2	The application shall display jobs and their estimated duration on a timeline	C
				R-4.1.3	The application needs a filter so that you can specify what data you want to view	B
US-5	Multiple tables	UC-5.1	Multiple table support	R-5.1.1	The application needs to be able to connect to multiple tables	A
				R-5.1.2	The application needs to show the different statuses of the tables	A
				R-5.1.3	The application needs to be able to securely store table information locally on the computer	B
		UC-5.2	Scan for multiple tables	R-5.2.1	The application shall allow the user to scan for multiple tables on a class C network	C
US-6	Job statistics report	UC-6.1	Generate production statisticts	R-6.1.1	The application needs to request the appropriate data from the HUB to generate a daily, weekly and monthly report	B
		UC-6.2	Display production report	R-6.2.1	The application needs to display the different time report in a way that maximizes readability	C
US-7	Type of application	UC-7.1	Handheld device	R-7.1.1	The application should both work on a handheld device (Android) and windows	B

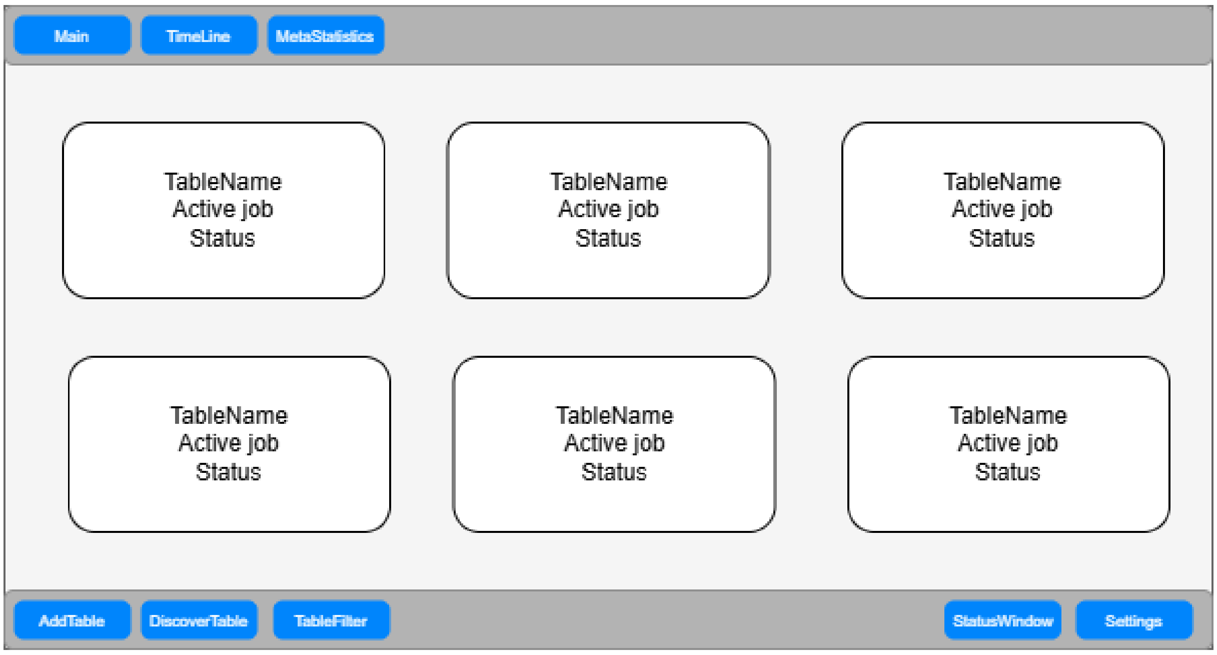
D Task System Requirements

Requirement ID	Requirement	Linked Items
KTT-110	R-1.1.1	KTT-100, KTT-99, KTT-98, KTT-159, KTT-171, KTT-178, KTT-184, KTT-233, KTT-237
KTT-111	R-1.1.2	KTT-191, KTT-225, KTT-193
KTT-112	R-1.2.1	KTT-192, KTT-198, KTT-200, KTT-268, KTT-194, KTT-228
KTT-113	R-1.2.2	KTT-268
KTT-119	R-2.1.1	KTT-190, KTT-257
KTT-120	R-2.2.1	KTT-189
KTT-121	R-2.3.1	KTT-298
KTT-125	R-3.1.1	KTT-168, KTT-185, KTT-258, KTT-243, KTT-252, KTT-284
KTT-127	R-3.2.1	KTT-193, KTT-224, KTT-240, KTT-245, KTT-248, KTT-255, KTT-275
KTT-128	R-3.2.2	
KTT-130	R-3.3.1	KTT-195, KTT-101
KTT-131	R-3.3.2	KTT-158, KTT-160, KTT-181, KTT-230, KTT-282, KTT-175
KTT-165	R-3.3.3	KTT-163, KTT-176, KTT-177, KTT-173, KTT-180, KTT-182, KTT-183, KTT-158, KTT-277
KTT-207	R-3.3.4	KTT-206, KTT-209, KTT-221, KTT-222, KTT-223, KTT-226, KTT-231, KTT-241, KTT-239, KTT-235, KTT-252, KTT-286, KTT-179, KTT-294, KTT-290, KTT-188, KTT-196, KTT-277, KTT-278, KTT-279, KTT-284, KTT-287, KTT-289, KTT-299, KTT-210, KTT-301, KTT-174, KTT-169
KTT-134	R-4.1.1	KTT-256
KTT-135	R-4.1.2	KTT-272, KTT-280
KTT-137	R-4.1.3	KTT-261, KTT-273

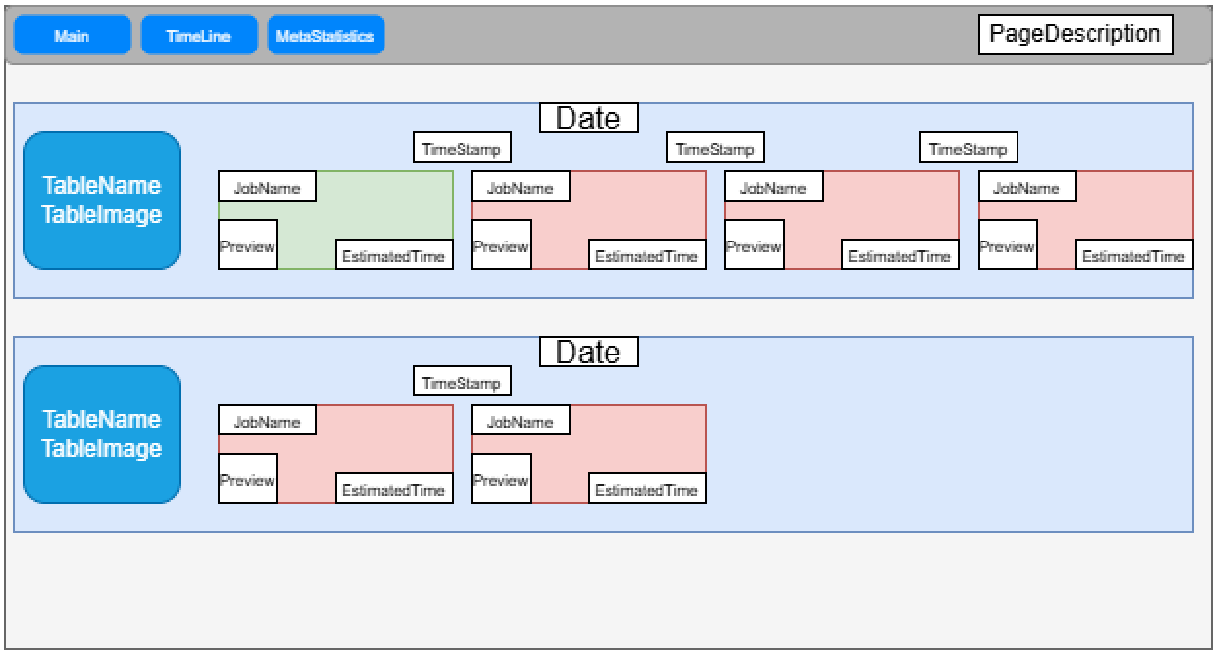
Requirement ID	**Requirement**	**Linked Items**
KTT-140	R-5.1.1	KTT-208, KTT-210
KTT-141	R-5.1.2	KTT-268
KTT-251	R-5.1.3	KTT-270, KTT-242
KTT-250	R-5.2.1	KTT-260
KTT-150	R-6.1.1	KTT-262, KTT-267, KTT-292
KTT-151	R-6.2.1	KTT-281, KTT-293, KTT-310, KTT-304, KTT-302, KTT-223, KTT-287, KTT-289, KTT-299

E GUI Mockups

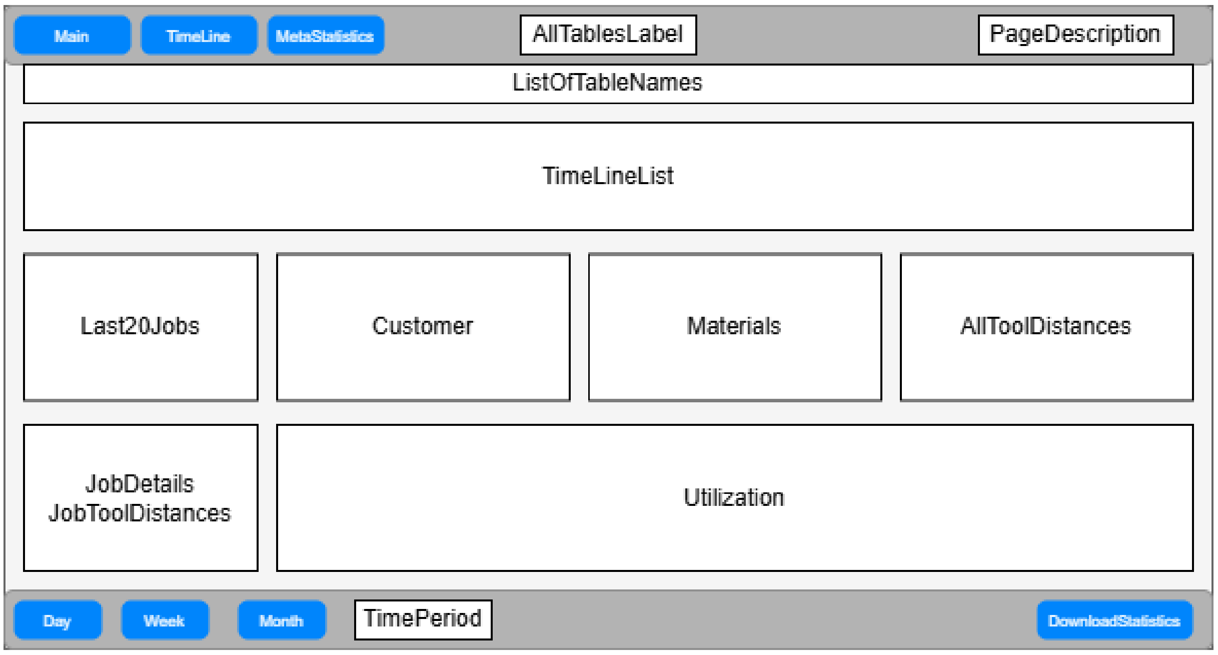
MainPage


Main
TimeLine
MetaStatistics
TableName Active job Status
TableName Active job Status
TableName Active job Status
TableName Active job Status
TableName Active job Status
TableName Active job Status
AddTable
DiscoverTable
TableFilter
StatusWindow
Settings

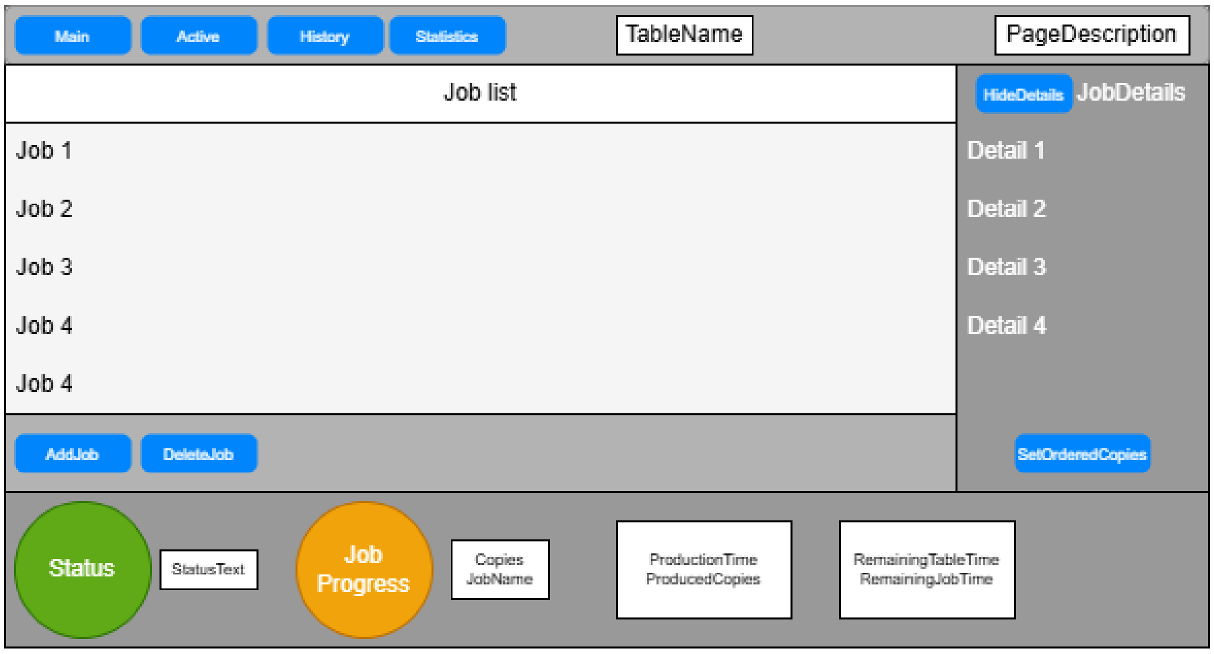
TimeLinePage



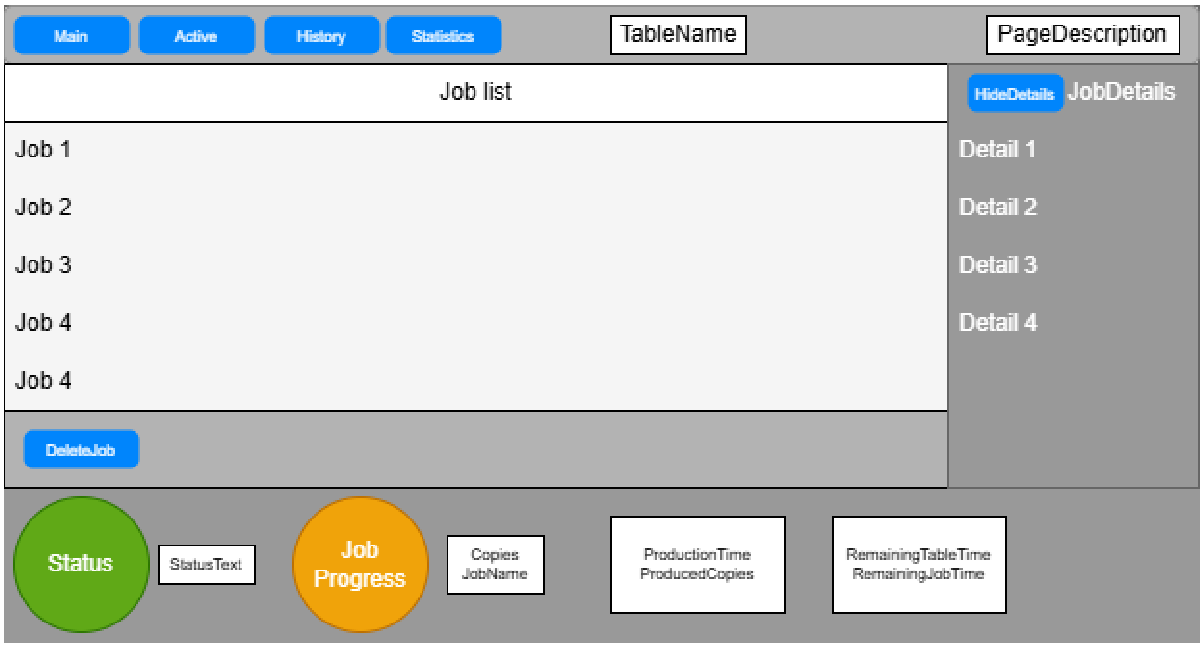
MetaStatisticsPage



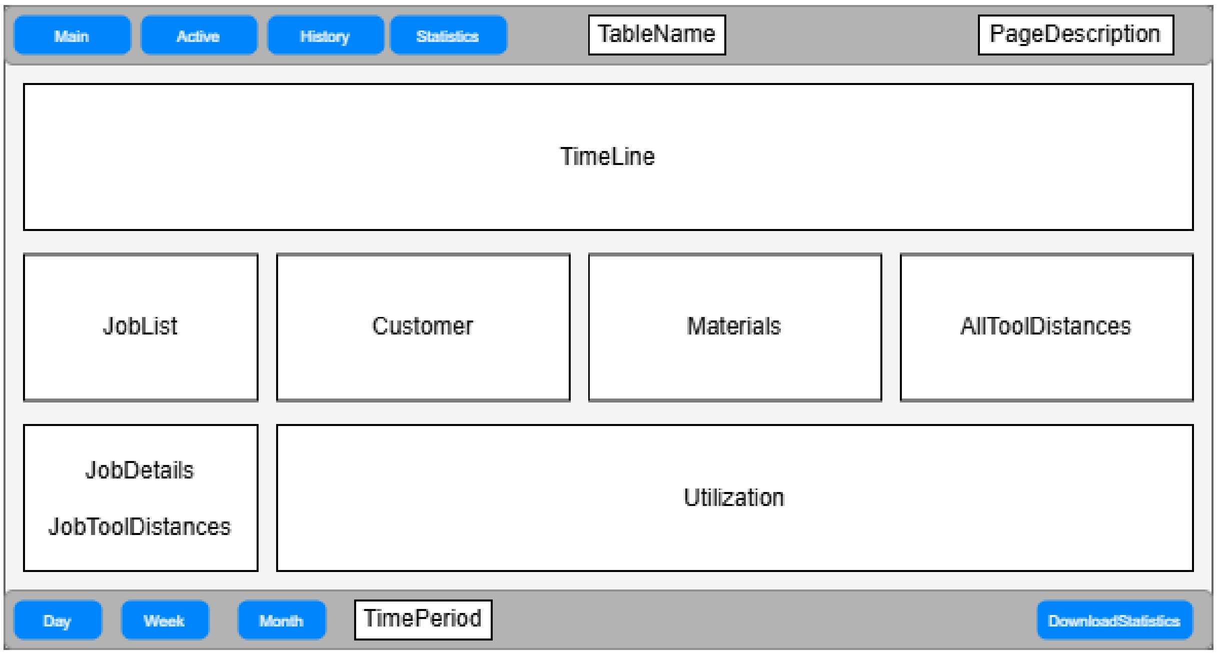
ActivePage



HistoryPage



StatisticsPage



F Detailed View Description

Overview of TimeLinePage

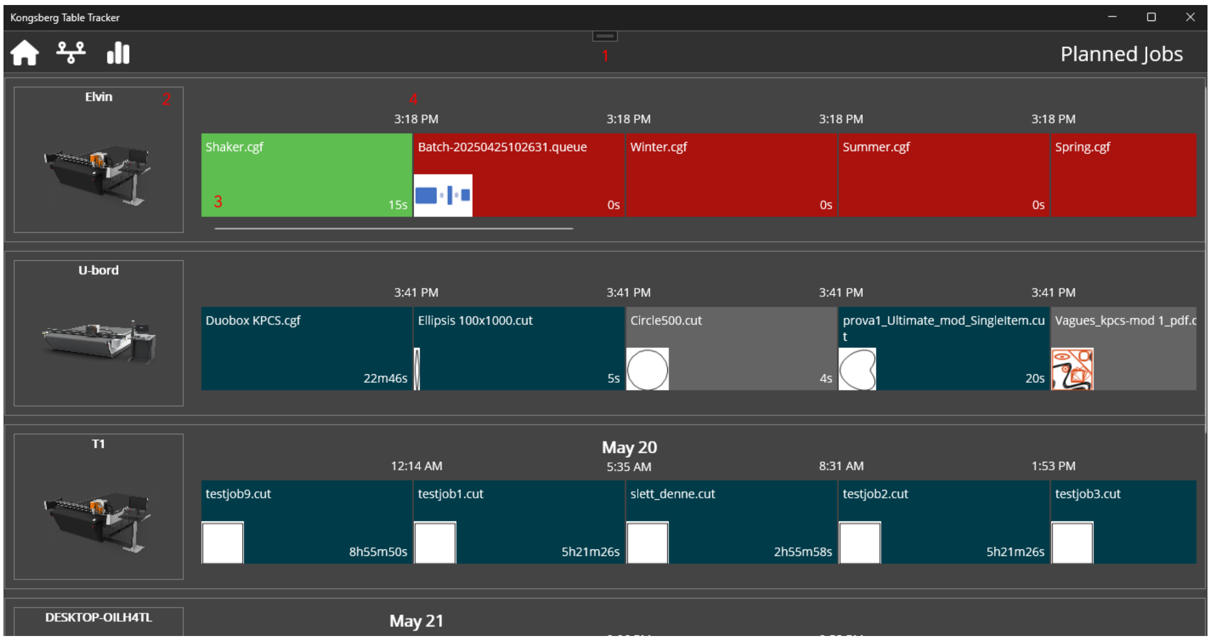


Figure 27: TimeLinePage screenshot with annotated UI elements

1. **Navigation bar** – Contain navigation buttons and page description.
2. **Table** – Table image and name. Each table has one dedicated row.
3. **Job** – Job item, with name and time duration.
4. **Estimated finished** – Estimated time of when the job is expected to finish.

Structure

The timeline page is built on a Grid with two rows. The first row is reserved for the navigation bar. The rest is allocated to a list of tables with their jobs.

Code-behind

Listing 9: TimeLine code-behind

```
using KongsbergTableTracker.ViewModels;

namespace KongsbergTableTracker.Views;

public partial class TimelinePage : ContentPage
{
    public TimelinePage(TimelineViewModel timelineViewModel)
 {
  InitializeComponent();
  BindingContext = timelineViewModel;
 }

    protected override void OnAppearing()
    {
        base.OnAppearing();
  if (BindingContext is TimelineViewModel timelineViewModel)
  {
   timelineViewModel.OnPageAppearing();
  }
    }

 protected override void OnDisappearing()
 {
  base.OnDisappearing();
        if (BindingContext is TimelineViewModel timelineViewModel)
        {
            timelineViewModel.OnPageDisappearing();
        }
    }
}
```

Overview of MetaStatisticsPage

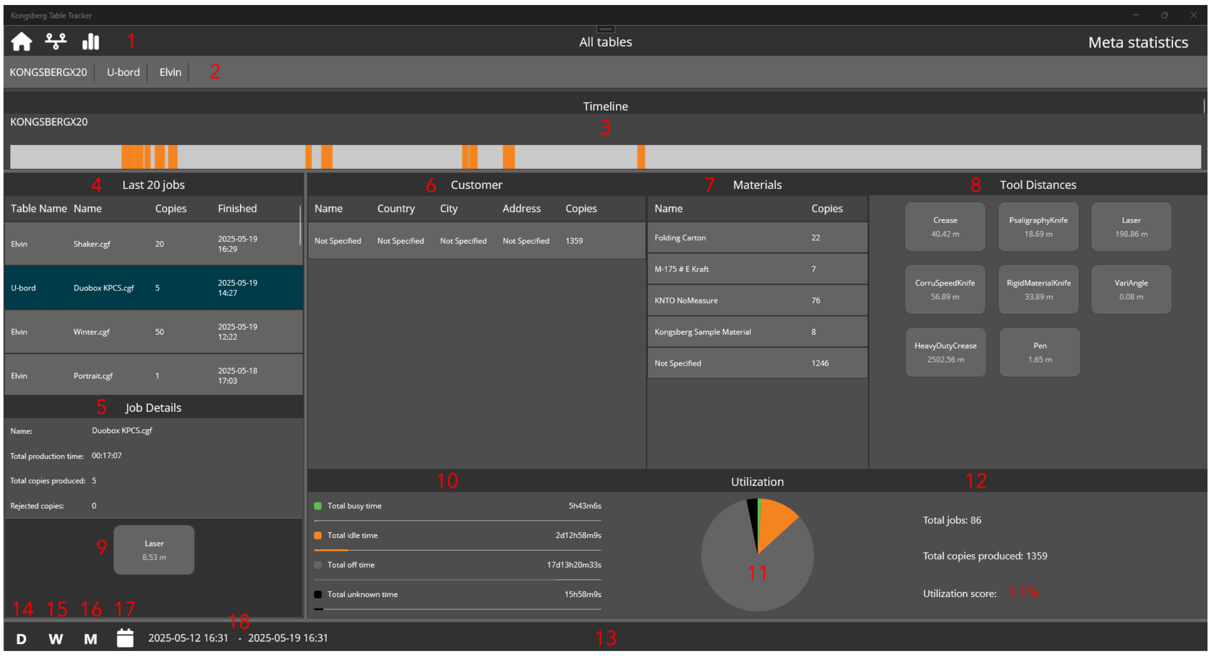


Table Name	Name	Copies	Finished
Elvin	Shaker.cgf	20	2025-05-19 16:29
U-bord	Duobox KPCS.cgf	5	2025-05-19 14:27
Elvin	Winter.cgf	50	2025-05-19 12:22
Elvin	Portrait.cgf	1	2025-05-18 17:03

Name:	Duobox KPCS.cgf
Total production time:	00:17:07
Total copies produced:	5
Rejected copies:	0

Laser
8.53 m

Name	Country	City	Address	Copies
Not Specified	Not Specified	Not Specified	Not Specified	1359

Name	Copies
Folding Carton	22
M-175 # E Kraft	7
KNTO NoMeasure	76
Kongsberg Sample Material	8
Not Specified	1246

Crease	40.42 m
PsaligraphyKnife	18.69 m
Laser	198.86 m
CorruSpeedKnife	56.89 m
RigidMaterialKnife	33.89 m
VariAngle	0.08 m
HeavyDutyCrease	2502.56 m
Pen	1.65 m

🟩 Total busy time	5h43m6s
🟧 Total idle time	2d12h58m9s
⬜ Total off time	17d13h20m33s
⬛ Total unknown time	15h58m9s

Total jobs: 86

Total copies produced: 1359

Utilization score: 1.1%

2025-05-12 16:31 - 2025-05-19 16:31

Figure 28: MetaStatisticsPage screenshot with annotated UI elements

1. **Navigation bar** – Navigation bar and page description.
2. **Tables** – A list of tables that have been collected data from.
3. **Timeline** – A scrollView of all the different timelines for each table. Timeline displays the different jobs that have been completed.
4. **Job list** – A list of the last 20 jobs from all tables.
5. **Job details** – Shows details about selected job (dark blue highlight).
6. **Customer** – Displays a list of all customers and how many customer is registered on each.
7. **Material** – Displays a list of used materials, and how many copies for each.
8. **Tool distances** – Display tool distance for all tables on all jobs.
9. **Tool distance** – Display tool distance for a single selected job.
10. **Table Utilization** – Display total time all tables have spent in the different states.
11. **Utilization chart** – Pie chart of table utilization.
12. **Production info** – Display total jobs, total produced copies and total utilization score from all tables.
13. **Task bar** - Contains buttons used on the page.
14. **Day** - Collect statistics for 24 hours back in time.

15. **Week** - Collect statistics from for the week.
16. **Month** - Collect statistics from the last month.
17. **Custom time** - Opens a popup where the user can select before and after date and time to collect statistics from.
18. **Time period** - Shows what time periode the statistics displayed is collected from.

Structure

MetaStatisticsPage is structured with multiple nested grids. The main grid concists of five rows and two columns, where the right column is set to be three times the size of the left column. The first row is reserved for the navigation bar. The second display the list of tables that have been used to collect statistics. On row three is the timeline for all the tables in the list above. All previous rows have spanned accross both columns, but not for row four. The left column in this row contains the job list and detials. The right column is a new nested grid. It contains customer, materials, tool distances and utilization. The bottom row is the task bar where the time periode for the statistics collected is decided.

Code-behind

The code-behind for the MetaStatisticsPage is clean, simple and straight forward. It takes in a MetaStatisticsViewModel as constructor parameter and sets the binding context to said ViewModel. This is what could be called a textbook example of a code-behind with mvvm is architecture. For the following examples, the code-behind will not be explained, as it is mostly similar.

Listing 10: MetaStatisticsPage code-behind

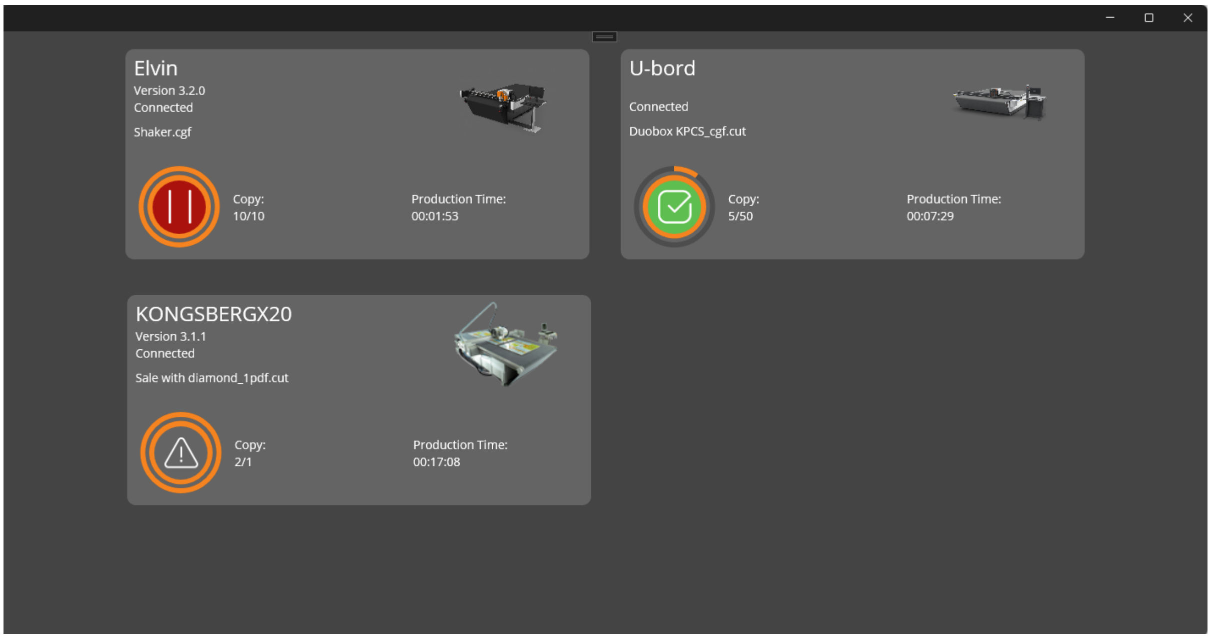
```
using KongsbergTableTracker.ViewModels;

namespace KongsbergTableTracker.Views;

public partial class MetaStatisticsPage : ContentPage
{
 public MetaStatisticsPage(MetaStatisticsViewModel metaStatisticsViewModel)
 {
        InitializeComponent();
        BindingContext = metaStatisticsViewModel;
    }
}
```

Overview of StatusPage

The StatusPage page is mostly similar to the MainPage. The differance is that it has no functionaloty, meaning the navigation bar, task bar and other functionality is removed or diabled. This page is purely for displaying purposes, so that the user can use the app on one screen, and display the status of all the tables on a seperate screen.



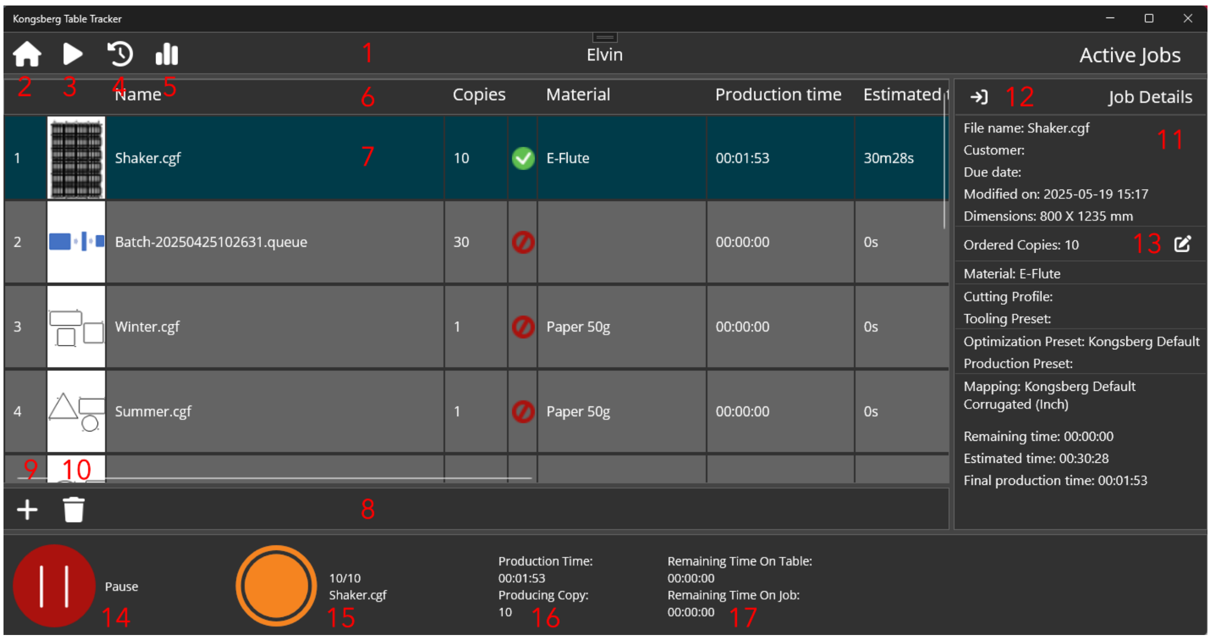
Elvin
Version 3.2.0
Connected
Shaker.cgf
Copy: 10/10
Production Time: 00:01:53

U-bord
Connected
Duobox KPCS_cgf.cut
Copy: 5/50
Production Time: 00:07:29

KONGSBERGX20
Version 3.1.1
Connected
Sale with diamond_1pdf.cut
Copy: 2/1
Production Time: 00:17:08

Figure 29: StatusPage screenshot

Overview of ActivePage



	Name	Copies		Material	Production time	Estimated
1	Shaker.cgf	10	✓	E-Flute	00:01:53	30m28s
2	Batch-20250425102631.queue	30	⊘		00:00:00	0s
3	Winter.cgf	1	⊘	Paper 50g	00:00:00	0s
4	Summer.cgf	1	⊘	Paper 50g	00:00:00	0s

File name: Shaker.cgf
Customer:
Due date:
Modified on: 2025-05-19 15:17
Dimensions: 800 X 1235 mm
Ordered Copies: 10
Material: E-Flute
Cutting Profile:
Tooling Preset:
Optimization Preset: Kongsberg Default
Production Preset:
Mapping: Kongsberg Default Corrugated (Inch)
Remaining time: 00:00:00
Estimated time: 00:30:28
Final production time: 00:01:53

10/10
Shaker.cgf

Production Time:
00:01:53
Producing Copy:
10

Remaining Time On Table:
00:00:00
Remaining Time On Job:
00:00:00

Figure 30: ActivePage screenshot with annotated UI elements

1. **Navigation bar** – Navigation bar, now in table context.
2. **Home button** – Navigate to MainPage.
3. **Active button** – Navigate to ActivePage.
4. **History button** – Navigate to HistoryPage.
5. **Statistics button** – Navigate to StatisticsPage.
6. **Job list** – CollectionView of all jobs and categories sorted in a grid.
7. **Job** – A specific job item. Dark blue mean the job is selected.
8. **Task bar** – Container for page specific buttons.
9. **Add job** – Lets the user uplad a job (.zip).
10. **Delete job** – Deletes the selected job from the job list.
11. **Job details** – Details appear when job is selected. Shows details for the selected job.
12. **Hide details** – Lets the user hide the job details. Click a job item to make it re-appear.
13. **Set ordered copies** – Lets the user set ordered copies amount on the selected job.
14. **Table status** – Shows the current status of the table.
15. **Job status** – Shows current status and progression of the job(s).

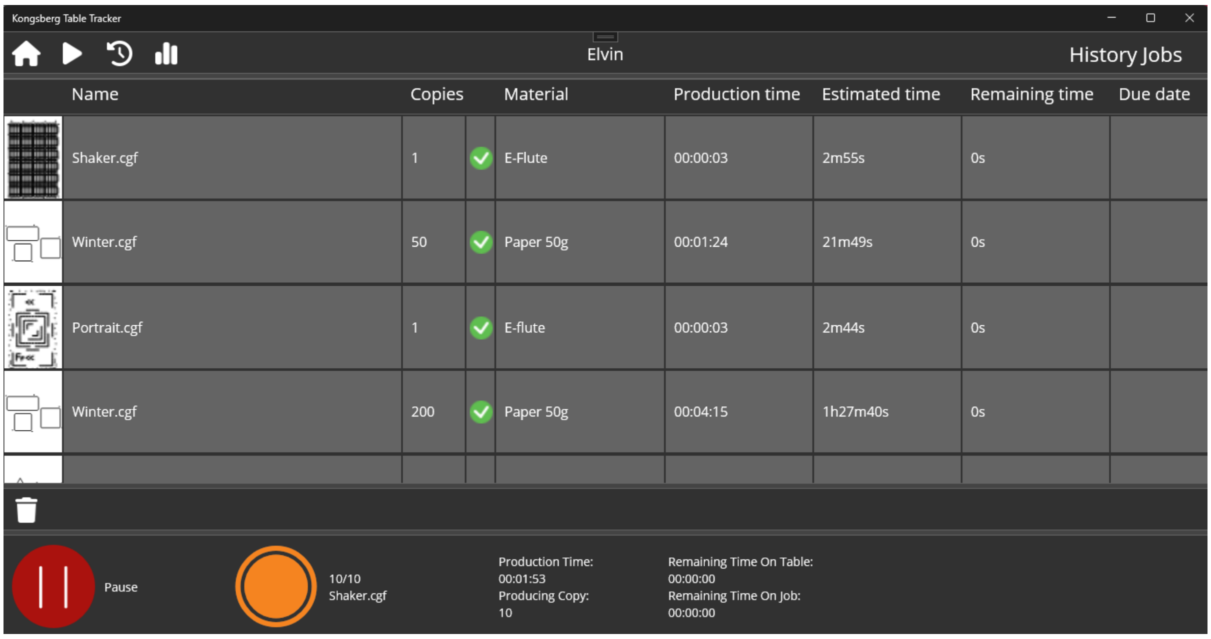
16. **Time and copies** – Show total job production time and current copy in production.

17. **Time details** – Shows estimated time left on table and job.

Structure

ActivePage is built with grid as the main layout. It has four rows and two columns. The top row is reserved for the navigation bar. The job list is located in the second row, left column. The JobDetails are located in the right column, and also span into the row below. The task bar is in row three, left column. When the detail panel is hidden, the right column gets the width changed to 0, therefore the left column recives the entire width of the page. The last row is where the InfoPanelContentView is located. It is defined in a separate XAML file, so that it can be used on both ActivePage and HistoryPage.

Overview of HistoryPage



Name	Copies		Material	Production time	Estimated time	Remaining time	Due date
Shaker.cgf	1	✓	E-Flute	00:00:03	2m55s	0s	
Winter.cgf	50	✓	Paper 50g	00:01:24	21m49s	0s	
Portrait.cgf	1	✓	E-flute	00:00:03	2m44s	0s	
Winter.cgf	200	✓	Paper 50g	00:04:15	1h27m40s	0s	

10/10
Shaker.cgf

Production Time:
00:01:53
Producing Copy:
10

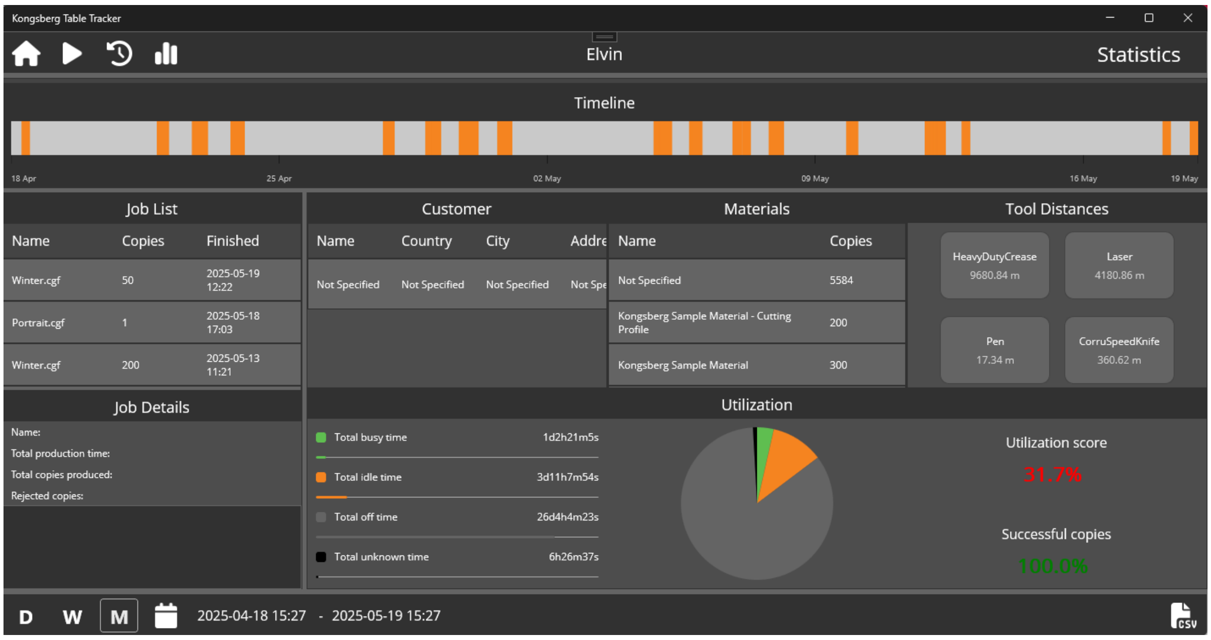
Remaining Time On Table:
00:00:00
Remaining Time On Job:
00:00:00

Figure 31: HistoryPage screenshot

Structure

HistoryPage is similar to ActivePage, but with some adjustments. The job list is of completed jobs. There is no button for adding jobs or setting copies, since this is a list of ”what has been”. There is also some changes to the columns, where the SortNumber (ID) is removed. There is also an additional column, although not visible here, that display the finished time for the job. Figure 31 also shows how both pages look with the job details hidden.

Overview of StatisticsPage



Job List

Name	Copies	Finished
Winter.cgf	50	2025-05-19 12:22
Portrait.cgf	1	2025-05-18 17:03
Winter.cgf	200	2025-05-13 11:21

Customer

Name	Country	City	Addre
Not Specified	Not Specified	Not Specified	Not Spe

Materials

Name	Copies
Not Specified	5584
Kongsberg Sample Material - Cutting Profile	200
Kongsberg Sample Material	300

Tool Distances

HeavyDutyCrease	9680.84 m
Laser	4180.86 m
Pen	17.34 m
CorruSpeedKnife	360.62 m

Job Details

Name:
Total production time:
Total copies produced:
Rejected copies:

Utilization

🟩 Total busy time	1d2h21m5s
🟧 Total idle time	3d11h7m54s
⬜ Total off time	26d4h4m23s
⬛ Total unknown time	6h26m37s

Utilization score
31.7%

Successful copies
100.0%

Figure 32: StatisticsPage screenshot

Structure

StatisticsPage is structured in the same way as MetaStatisticsPage, but is now in a table context. That means it only concers one table. Notice that the list of tables is gone. There is also an additional button on the right side on the task bar (bottom right). This is a button that let's the user download a .csv file of the currently displayed time periode. Also notice that no job is selected at the moment, therefore there is no tool distance in the bottom left pane. It will only display when a job from the list is selected.

Overview of Popups

The KTT application contains many popups from the CommunityToolKit library. They mostly concists of two parts, the header and body. They are explained in the pages where they are relevant above. Under are screenshots of each of them.

AddTablePopup

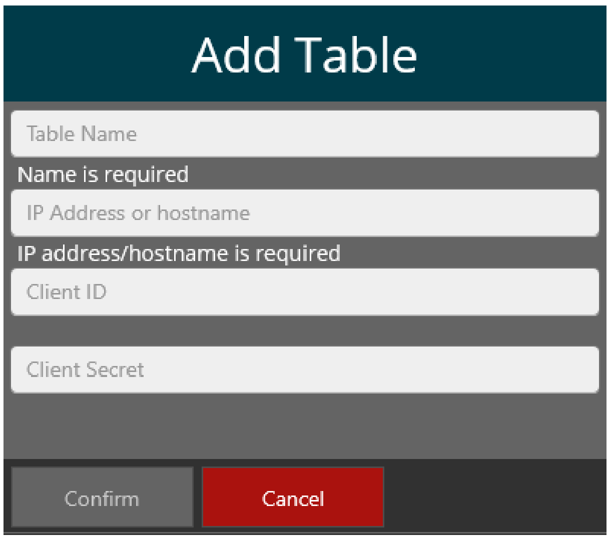
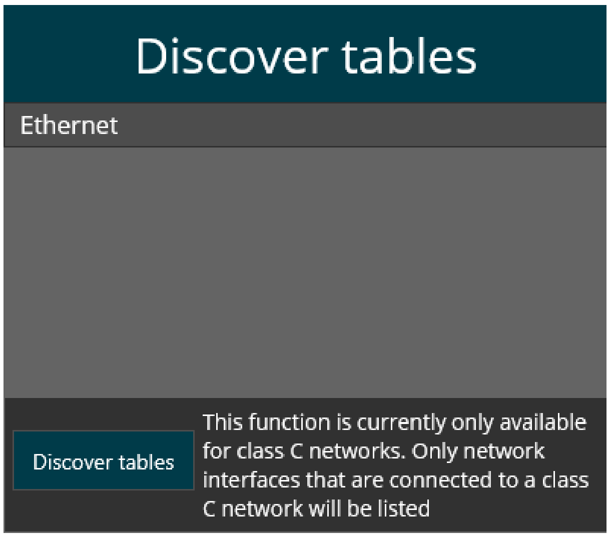


Figure 33: AddTable popup screenshot

The button on the AddTablePopup is gray because required fields are empty. The Edit-TablePopup looks the same as AddTablePopup, but fills the filds with information stored in the table item on appearing and has a different header.

DiscoverTablePopup



This function is currently only available for class C networks. Only network interfaces that are connected to a class C network will be listed

Figure 34: DiscoverTable popup screenshot

TableFilterPopup

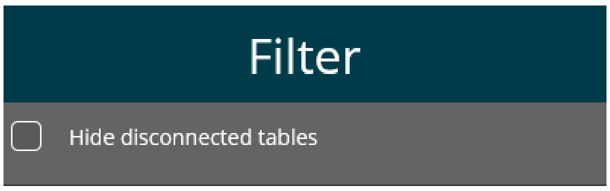
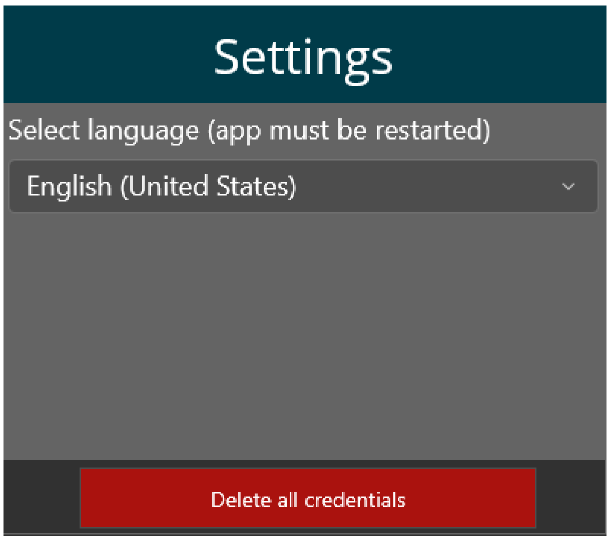


Figure 35: TableFilter popup screenshot

SettingsPopup



Select language (app must be restarted)

English (United States)

Figure 36: Settings popup screenshot

CustomStatisticsPopup

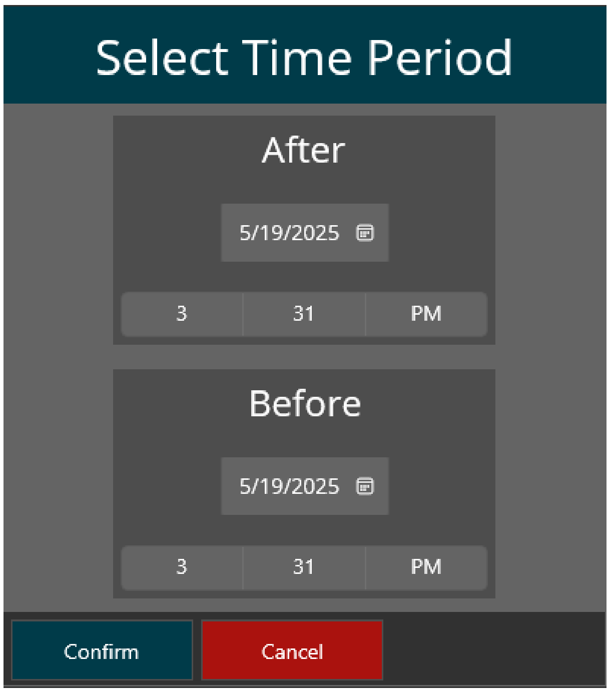


Figure 37: CustomStatistics popup screenshot

ColumnSelectorPopup

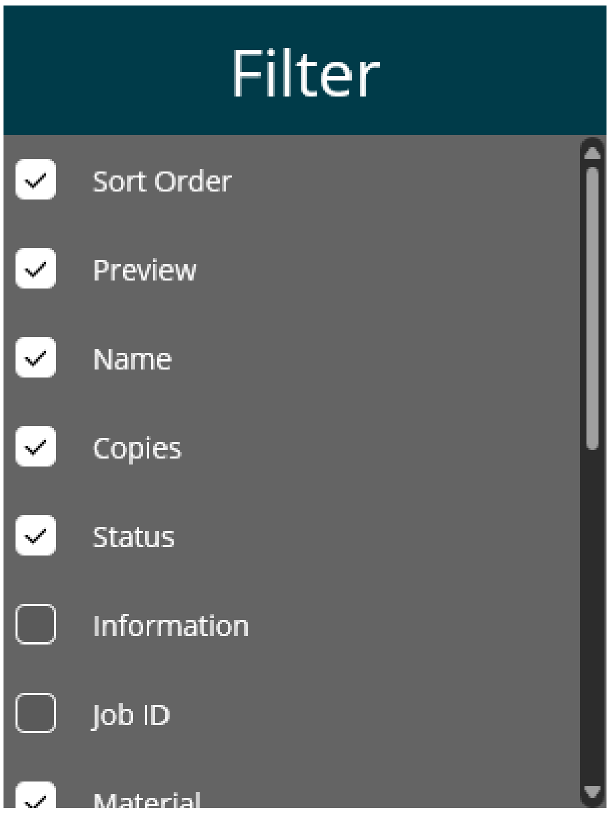


Figure 38: ColumnSelector popup screenshot

G Unit Testing Report

Unit Testing

Introduction

Unit testing is a method used to check that small parts of an application work the right way. Each test looks at one method or function and checks the result based on what you give it. The goal is to find bugs early, make the system more stable, and help developers make changes safely. (more info here..)

In this project, I used unit testing to check important features that talk to the backend and handle different parts of the system. I used xUnit in Visual Studio to write our tests, and I used 'Fact' to mark each scenario. The reason I say scenarios is because inside one test class I can have multiple 'fact' attributes. The 'fact attribute tells the system "This is a unit test! ". They are different test scenarios for the function, in other words different inputs to the function. So to fully test a function I need to have multiple 'fact' attributes depending on the function so I can test all the different inputs, this gives me the outputs for the different inputs and I can see if they failed or not in the test explorer where I run the tests.

If the test fails

So if an input gives a unexpected output I can check the generated error message. For example one time I gave 3.14 as an expected output and I got 3,14 back with a comma, this is because I had my computer set to Norwegian so I just needed to change it to a comma. And this is good because the logic of the function worked and passed. But sometimes the logic of the function does not work as expected and then we have to find that function and fix it right away.

How I organized the tests

To organize the tests, I set up multiple test environments, one for models, one for services and so on. And inside the models test environment I have folders of the classes and inside there I have the functions in the classes I test. So, for example I want to test a function called 'ConvertValue()'. That function is located In Models -> TableModel. So this is how the file structure looks like in the test environment also. This makes it very easy for us to find a specific test.

I followed the Arrange-Act-Assert (AAA) format. First, I set up things (Arrange), then ran the method (Act), and finally checked the result (Assert).

Why We Used Mocks

To test methods that talk to the backend, I used mock objects to fake the server. This way, I didn't need to connect to a real server. I used a class called FakeHttpMessageHandler to return fake HTTP responses, so I could test success, failure, or errors.

I also made a class called ApiServiceWithClient. It is based on ApiService, but we changed it to let us set the HttpClient from outside. This helped us test how the API service works without changing the main code.

Another class I used was FakeApiService. It helped us test login and security. I used it to return a fake token so I could test logic that depends on being logged in, without calling a real server.

These mocks helped us test only the parts of code we wanted. The tests ran fast, didn't depend on the network, and I could run them again and again with the same result.

Why These Functions Were Tested

The **GetActiveJobs** method was tested because it reads job data from a backend API. It is important that this information is read correctly.

The **CheckAuthentication** method was tested because it checks if the user has permission. This keeps the system safe.

The **GenerateLocalIpList** method looks for devices on the network. I tested it to make sure it finds only the right devices.

The **RequestAccessToken** method sends the client ID and secret to get a token. This token is used to login. I tested this to make sure it works for both good and bad inputs.

The **FindProperty** method was tested because it reads data from a list of properties and tries to turn it into the correct type, like a number or an enum. I tested it to make sure it returns the right value and doesn't crash when something is missing or wrong.

The **ConvertValue** method was tested because it takes a string and a data type and tries to return a real value like an integer, double, boolean, or enum. I tested different types and inputs, including wrong ones, to make sure the method gives the correct result or a safe default when needed.

1.0 - GetActiveJobsTest.cs

TaskID KTT254-UNITTEST/APIServiceTests/GetActiveJobsTest.cs

Purpose

The GetActiveJobs method in the ApiServiceWithClient class retrieves active job data from a REST API and returns it as a structured object.

These tests aim to:

* Confirm correct parsing of valid JSON responses from the API.
* Ensure that null API responses are handled appropriately.

Testing Technique

Structure:

* Arrange: A fake HTTP response is set up using the FakeHttpMessageHandler.
* Act: The GetActiveJobs method is called.
* Assert: The output is validated to ensure it matches expectations.

Test Cases

- Scenario 1: API returns a valid JobQueue

* Setup: A JSON response with one job is simulated.
* Expected Result: The returned object should not be null. The JobTasks list should contain one item with expected ID and name.
* Test Pass Criteria: The response is correctly deserialized and matches expected structure.

- Scenario 2: API returns "null"

* Setup: A plain JSON value "null" is returned from the simulated API.
* Expected Result: The method should return null.
* Test Pass Criteria: The application handles the null response without errors.

Importance of This Test

* Validates the method's ability to interact with external services and handle different response formats.
* Ensures stability and accuracy of job-related data consumed by the application.
* Helps identify issues in deserialization or incorrect assumptions about API data.

Test Result

Both tests pass using the custom HTTP handler, confirming that the method performs correctly with valid and null API responses.

Lessons Learned

* Custom HTTP handlers are effective for testing HTTP client behavior.
* Including tests for edge cases like null values improves reliability.
* Isolated, focused unit tests make the codebase more maintainable over time.

2.0 - CheckAuthenticationTests.cs

TaskID: KTT266-UNITTEST/APIServiceTests/CheckAuthenticationTests.cs

Purpose

The CheckAuthentication method in the ApiServiceWithClient class validates user credentials by communicating with the backend API.

These tests aim to:

* Ensure that the method returns true for valid credentials.
* Ensure that the method returns false or handles the result correctly for invalid or unauthorized access attempts.

Testing Technique

Structure:

* Arrange: A simulated HTTP response is configured with a specific status code or content.
* Act: The CheckAuthentication method is invoked using this fake setup.
* Assert: The returned result is evaluated for correctness (true or false).

Test Cases

- Scenario 1: API returns success for valid credentials

* Setup: FakeHttpMessageHandler is configured to return an HTTP 200 OK response.
* Expected Result: The method should return true.
* Test Pass Criteria: The application correctly interprets successful authentication.

- Scenario 2: API returns unauthorized for invalid credentials

* Setup: A 401 Unauthorized response is simulated.
* Expected Result: The method should return false.
* Test Pass Criteria: The method handles the authentication failure.

Importance of This Test

* Authentication is a critical security function and must be validated under different conditions.
* Ensures users cannot access the system with incorrect credentials.
* Helps prevent false positives or negatives in authentication logic.

Test Result

Tests confirm that the CheckAuthentication method accurately interprets both valid and invalid API responses.

Lessons Learned

* Simulating both success and failure scenarios is essential for robust authentication logic.
* Fake HTTP handlers provide a simple yet effective method for isolating the logic from real backend systems.
* Clear assertion logic improves test readability and reliability.

3.0 - GenerateLocalIpListTEST.cs

Path: KTT264-UNITTEST/TableScanServiceTests/GenerateLocalIpListTEST.cs

Purpose

The GenerateLocalIpList method in the TableScanService class scans a local subnet range and identifies reachable IP addresses where table devices may be hosted.

These tests aim to:

* Ensure that the method returns a complete and accurate list of reachable IPs.
* Confirm that unreachable or invalid addresses are excluded from the result.

Testing Technique

Approach:

A custom test double or simulation is used in place of actual network calls. This avoids depending on real network availability while preserving the logic of iterating IP ranges.

Structure:

* Arrange: A set of mock network responses is prepared.
* Act: The GenerateLocalIpList method is executed.
* Assert: The method is verified to return only reachable IP addresses.

Test Cases

- Scenario: Simulated subnet with mixed reachable/unreachable IPs

* Setup: Simulate a subnet where only some IPs respond.
* Expected Result: The method should only include the reachable IPs.
* Test Pass Criteria: The returned list matches the simulated available addresses.

Importance of This Test

* This method underpins the discovery of table devices on the local network.
* Verifying its correctness is essential to ensure that all available devices are detected reliably.
* Helps identify potential issues in IP filtering or timeout logic.

Test Result

The method correctly returns the expected list of reachable IP addresses based on the simulation.

Lessons Learned

* Validating network-dependent logic in isolation helps improve reliability and test coverage.
* Designing for testability in service methods leads to cleaner architecture.
* Mocking or faking system calls is a powerful technique in integration-level testing.

4.0 – RequestAccessToken.cs

Path: KTTUnittesting/APIServiceTests/RequestAccessTokenTests.cs

Purpose

This method is responsible for authenticating a client using a client ID and client secret, and returning an access token received from the backend API.

These tests aim to:

* Ensure the method returns null if the ClientId or ClientSecret is missing or invalid.
* Verify that a valid token is returned when credentials are correct and the API responds with success.
* Confirm that the method handles unauthorized responses correctly by returning null when the API denies access (401 Unauthorized).

Testing Technique

Approach:

Tests were written using the xUnit framework with [Fact] attributes. The method's backend communication was simulated using a FakeHttpMessageHandler to control the HTTP response.

Structure:

* Arrange: Configure a fake HTTP response with desired status and content.
* Act: Call the RequestAccessToken method with different inputs.
* Assert: Validate whether the result is null or a valid TableAuthentication object depending on the scenario.

Test Cases

Scenario 1: Missing or invalid input
* Setup: Use null, empty, or whitespace values for ClientId or ClientSecret.
* Expected Result: Method returns null.
* Test Pass Criteria: No HTTP call is made, and the method exits early.

Scenario 2: Valid credentials and successful response
* Setup: Fake a 200 OK response with a valid JSON access token.
* Expected Result: A TableAuthentication object is returned with correct values.
* Test Pass Criteria: Deserialization and return of the token object succeeds.

Scenario 3: Invalid credentials (401 Unauthorized)
* Setup: Fake a 401 Unauthorized response.
* Expected Result: Method catches the exception and returns null.
* Test Pass Criteria: No unhandled exception; method handles the case cleanly.

Importance of This Test

* Verifies the app can correctly authenticate with the backend using secure credentials.
* Ensures users without proper credentials are denied access.
* Confirms the application handles failed authentication attempts gracefully.

Test Result

All three key scenarios pass. The method returns correct values for valid input, handles unauthorized access as expected, and validates inputs up front.

Lessons Learned

* Testing both positive and negative paths helps ensure the method is reliable and secure.
* Early input validation prevents unnecessary backend calls.
* Controlled HTTP simulation

5.0 – FindPropertyTests.cs

Task ID: KTT276-UNITTEST/TableModelTests/FindPropertyTests.cs

Purpose

The FindProperty<T> method in the TableModel class retrieves a value from the model's Properties list by matching a given name and converting its string value to the expected type T. This method supports various types, including strings, booleans, and enums such as TableStatuses.

These tests aim to:
* Verify that the method returns null when the Properties list is missing.
* Ensure the method handles unmatched property names correctly.
* Confirm that the method returns null if a property's value is not set.
* Validate that the method converts valid string values to the correct type.

Testing Technique

Structure:
* Arrange: A TableModel is created with either a valid or null Properties list.
* Act: The FindProperty<T> method is invoked with a specific Name and type.
* Assert: The returned result is checked for correctness (expected value or null).

Test Cases

Scenario 1: Properties list is null
* Setup: A TableModel is created with its Properties list set to null.
* Expected Result: The method returns null.
* Test Pass Criteria: The method handles the missing list safely without throwing an exception.

Scenario 2: No matching property found
* Setup: The list contains a property with a different name than the one requested.
* Expected Result: The method returns null.
* Test Pass Criteria: The method does not return unrelated or incorrect data.

Scenario 3: Property value is null
* Setup: A property with the correct name exists, but its Value is null.
* Expected Result: The method returns null.
* Test Pass Criteria: No conversion is attempted; the method returns the default value safely.

Scenario 4: Valid property is found and converted
* Setup: A property with name Status, value "Busy", and data type TableStatuses is added.
* Expected Result: The method returns the enum value TableStatuses.Busy.
* Test Pass Criteria: The string is correctly parsed and returned as the expected enum type.

Importance of This Test

* Ensures that TableModel can safely handle missing, incomplete, or incorrect data from the backend.
* Confirms the logic for string-to-type conversion works as intended across different types.
* Protects against potential runtime errors when accessing or interpreting properties dynamically.

Test Result

All four tests passed successfully. This confirms that the method behaves correctly for both edge cases and normal input scenarios.

Lessons Learned

* Testing null and non-matching conditions helps catch overlooked failure paths.
* Generic methods offer flexibility, but their behavior must be verified for different expected types.
* Including realistic and failure-based scenarios leads to more robust and maintainable code.

6.0 - FindPropertyTests.cs

TaskID: KTT-291-UNITTEST/Models/TableModel/FindPropertyTests.cs

Purpose

The FindProperty<T> method in the TableModel class retrieves a property by name from the table's Properties list and attempts to convert its value to the specified generic type. These unit tests are designed to:

* Confirm that the method returns null if a property's value is not set.
* Validate that the method converts valid string values to the correct type.

Testing Technique

Structure:

* Arrange: A TableModel is created with either a valid or null Properties list.
* Act: The FindProperty<T> method is invoked with a specific Name and type.
* Assert: The returned result is checked for correctness (expected value or null).

Test Cases

Scenario 1: Properties list is null

* Setup: A TableModel is created with its Properties list set to null.
* Expected Result: The method returns null.
* Test Pass Criteria: The method handles the missing list safely without throwing an exception.

Scenario 2: No matching property found

* Setup: The list contains a property with a different name than the one requested.
* Expected Result: The method returns null.
* Test Pass Criteria: The method does not return unrelated or incorrect data.

Scenario 3: Property value is null

* Setup: A property with the correct name exists, but its Value is null.
* Expected Result: The method returns null.
* Test Pass Criteria: No conversion is attempted; the method returns the default value safely.

Scenario 4: Valid property is found and converted

* Setup: A property with name Status, value "Busy", and data type TableStatuses is added.
* Expected Result: The method returns the enum value TableStatuses.Busy.
* Test Pass Criteria: The string is correctly parsed and returned as the expected enum type.

Importance of This Test

* Ensures that TableModel can safely handle missing, incomplete, or incorrect data from the backend.
* Prevents potential runtime errors due to invalid type conversions.
* Confirms that the method supports strong typing while working with dynamic data.
* Increases reliability of business logic and UI elements that depend on correctly typed property values.

Test Result

All test scenarios passed successfully, confirming that FindProperty<T> behaves as expected across null, missing, invalid, and valid data cases.

7.0 - ConvertValueTests.cs

TaskID: KTT-295-UNITTEST/Models/TableModel/ConvertValueTests.cs

Purpose

The ConvertValue<T> method in the TableModel class is responsible for converting a string and an associated DataType enum into a strongly typed value of type T. This unit test ensures that the method:

* Correctly converts supported data types (int, bool, double, string, enums, etc.).
* Gracefully returns default values when conversion fails.
* Does not throw exceptions for invalid inputs.

Testing Technique

Structure:

* Arrange: A TableModel instance is created using test-safe dummy dependencies.
* Act: The ConvertValue<T> method is called using reflection with different input values.
* Assert: The output is checked for correct type and expected value.

Test Cases

Scenario 1: Convert valid int

* Setup: DataType is Int32, value is "123".
* Expected Result: The method returns integer 123.
* Test Pass Criteria: The string is parsed and returned as an integer.

Scenario 2: Convert valid boolean

* Setup: DataType is Boolean, value is "true".
* Expected Result: The method returns boolean true.
* Test Pass Criteria: The string is parsed and returned as a boolean.

Scenario 3: Convert valid double

* Setup: DataType is Double, value is "3.14".
* Expected Result: The method returns double 3,14.
* Test Pass Criteria: Conversion respects system culture and returns expected numeric value.

Scenario 4: Convert valid enum

* Setup: DataType is TableStatuses, value is "Idle".
* Expected Result: The method returns TableStatuses.Idle.
* Test Pass Criteria: The enum string is parsed into the correct enum value.

Scenario 5: Invalid int conversion

* Setup: DataType is Int32, value is "abc".
* Expected Result: The method returns default int (0).
* Test Pass Criteria: Conversion fails without exception.

Importance of This Test

* Ensures that dynamic string values from APIs or serialized data can be safely and accurately interpreted.
* Verifies resilience of the conversion logic in the presence of malformed input.
* Supports stable behavior of higher-level methods relying on parsed property values.
* Helps prevent runtime crashes due to unhandled format exceptions.

Test Result

At first, I tried using a period for decimal numbers, but it didn’t work. Since my system is in Norwegian, I had to use a comma instead. All test cases passed successfully after that, confirming that ConvertValue handles localization correctly and returns the expected values.

8.0 - TimeSpanToDHMSConverterTests.cs

TaskID: KTT-296-UNITTEST/Utilities/Converters/TimeSpanToDHMSConverterTests.cs

Purpose

The TimeSpanToDHMSConverter class takes a TimeSpan object and converts it into a string that shows days, hours, minutes and seconds using short words like 'd', 'h', 'm' and 's'. If the value is not a TimeSpan, or if it is empty, it returns the text 'Unknown'. I tested this function to make sure it gives the correct string for different TimeSpan values.

Testing Technique

I wrote unit tests using the xUnit framework. I made a test helper called TimeSpanToDHMSConverterAccessor to call the Convert method with reflection. I also made fake AppResources values like AbbreviatedDay = 'd' to make the test work without touching the real code.

Test Cases

Scenario 1: The input is null. The expected result is 'Unknown'.

Scenario 2: The input is 10 seconds. The expected result is '10s'.

Scenario 3: The input is 1 minute and 15 seconds. The result should be '1m15s'.

Scenario 4: The input is 1 hour, 2 minutes and 3 seconds. The result should be '1h2m3s'.

Scenario 5: The input is 2 days, 3 hours, 4 minutes and 5 seconds. The result should be '2d3h4m5s'.

Scenario 6: The input is 2 years, 0 days,0 hours, 0 minutes, 0 seconds, the result should be ‘730d,0h,0m,0s'.

Importance of This Test

This test is important because it makes sure the converter shows the time in a nice and simple way for the user. If the result is wrong, the app could show confusing text or crash. It also makes sure that empty input returns 'Unknown'.

Test Result

All test cases passed. The converter gives the expected output for every TimeSpan tested. I did not test culture-specific formats because the values are hardcoded like 'd', 'h', etc.

Lessons Learned

I learned that using reflection is helpful when I cannot access the method directly. I also learned that making fake AppResources is useful when I cannot change the real app code.

H Regression Test

Regression Test for KTT

1.Content

2. Description

These tests are part of the regression testing for our application. The goal is to make sure that everything still works as expected after all development is finished. Each test checks a core part of the application to confirm that no bugs or issues have been introduced during the project. For example, we test if the system can still connect to the Kongsberg HUB, if it is possible to add new tables, and if the table statuses are shown correctly. We also check that job data appears with the correct estimated duration and that the operator gets the right information.

The tests include which requirements are being tested, what needs to be ready before the test starts, the steps the tester must follow, and what should happen if the test is successful. It also shows who performs the test and whether the result is a pass or a fail. These tests help us make sure that the application is stable and ready to be delivered.

3. Prerequisites

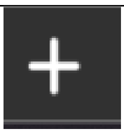
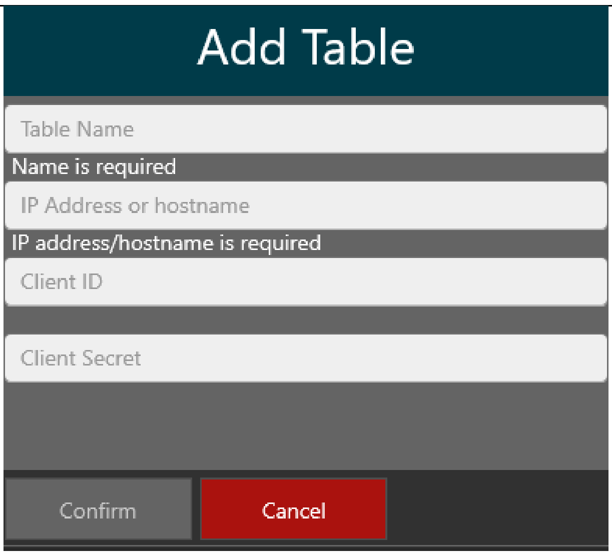
HUB is up and running and connected and the network connectivity is stable and we have valid credentials for the tables to connect later in the tests.

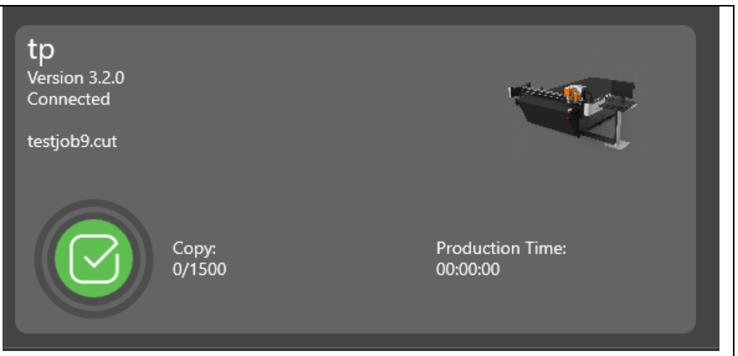
4.Testing

Test 1 – Establish connection and add tables

Requirements tested: R-1.1.1, R-1.1.2, R-3.3.3

Test ID: T-1

Steps	Test	Comment	Visual
1	Open Kongsberg Hub Client Configurator on the Kongsberg Table		
2	Click add table in the application and write in the IP and Hostname		
3	Write in the ClientID and the Client Secret.		 Add Table Table Name Name is required IP Address or hostname IP address/hostname is required Client ID Client Secret Confirm Cancel

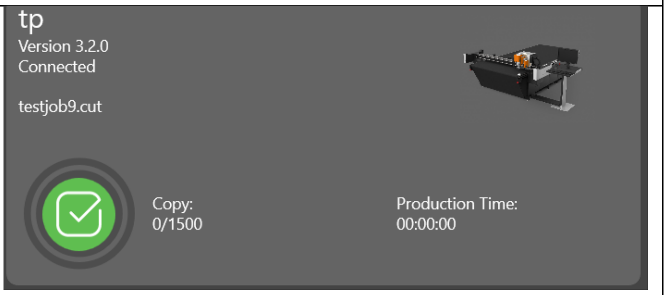
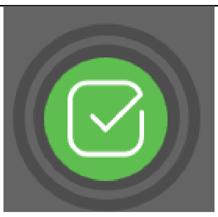
4	Click Confirm	Table appears in the GUI with correct information.	 tp Version 3.2.0 Connected testjob9.cut Copy: 0/1500 Production Time: 00:00:00

Tester	Magnus Trillhus Olsnes
Date	May 2025
Pass/Fail	Pass

Test 2 -Get table status

Requirements tested: R-1.1.2, R-1.2.1, R-5.1.2

Test ID: T-2

Steps	Test	Comment	Visual
1	Open the Home page on the application where the table(s) is/are displayed	You need to already have one or more tables connected.	 tp Version 3.2.0 Connected testjob9.cut Copy: 0/1500 Production Time: 00:00:00
2	Check if the status is displayed in an intuitive way, such as correct color and text	The table status is shown on the bottom left.	

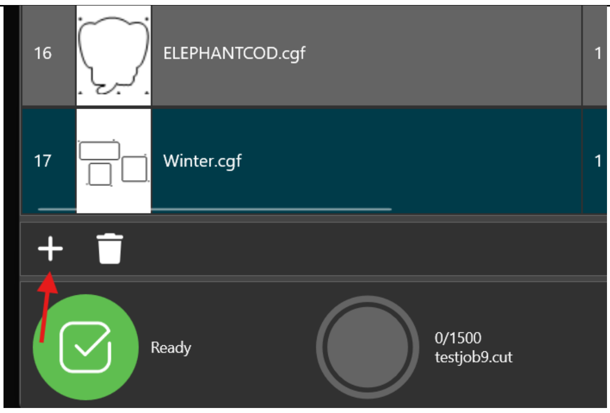
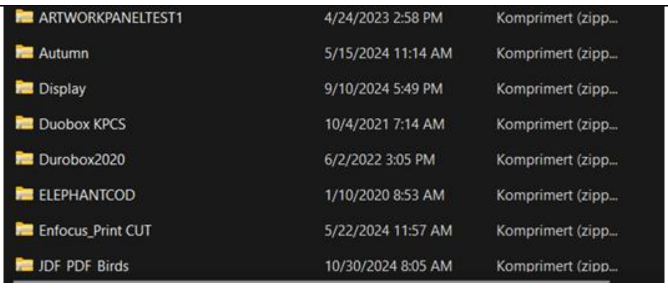
3	Click on a table	The statuses should match eachother	

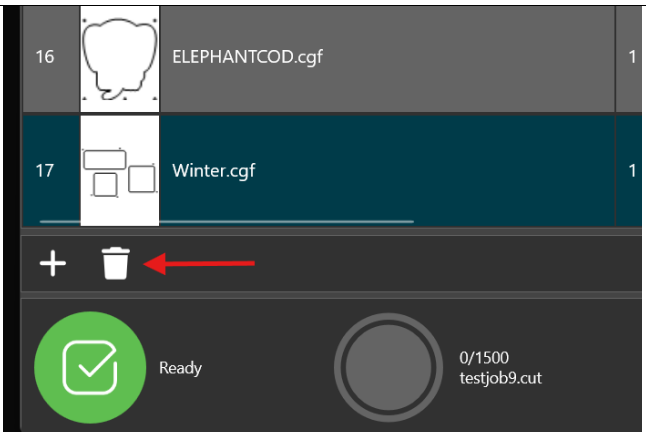
Tester	Magnus Trillhus Olsnes
Date	May 2025
Pass/Fail	Pass

Test 3 Upload job

Requirements tested: R-2.1.1, R-3.3.1

Test ID: T-3

Steps	Test	Comment	Visual
1	Click on a connected table, and then click the "add button" to manually add a job.	The file explorer should open and you can choose the job to upload	 16 ELEPHANTCOD.cgf 1; 17 Winter.cgf 1; Ready; 0/1500 testjob9.cut
2	2. Select a job from your local computer and upload it	ZIP file	 ARTWORKPANELTEST1 4/24/2023 2:58 PM Komprimert (zipp...; Autumn 5/15/2024 11:14 AM Komprimert (zipp...; Display 9/10/2024 5:49 PM Komprimert (zipp...; Duobox KPCS 10/4/2021 7:14 AM Komprimert (zipp...; Durobox2020 6/2/2022 3:05 PM Komprimert (zipp...; ELEPHANTCOD 1/10/2020 8:53 AM Komprimert (zipp...; Enfocus_Print CUT 5/22/2024 11:57 AM Komprimert (zipp...; JDF PDF Birds 10/30/2024 8:05 AM Komprimert (zipp...

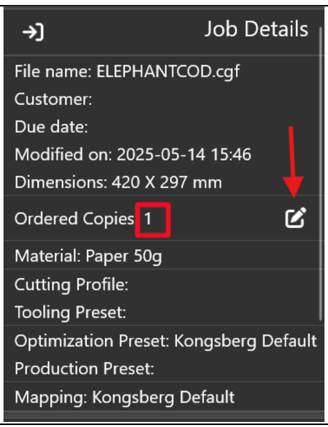
3	After adding a job, delete it by selecting it and click the delete button	Should disappear from the job list	 16 ELEPHANTCOD.cgf 1 17 Winter.cgf 1 Ready 0/1500 testjob9.cut

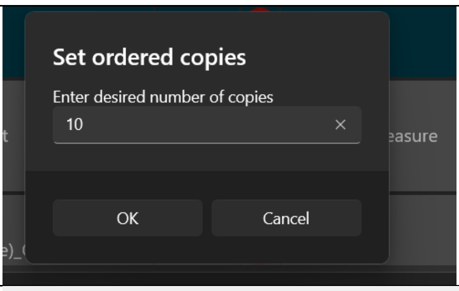
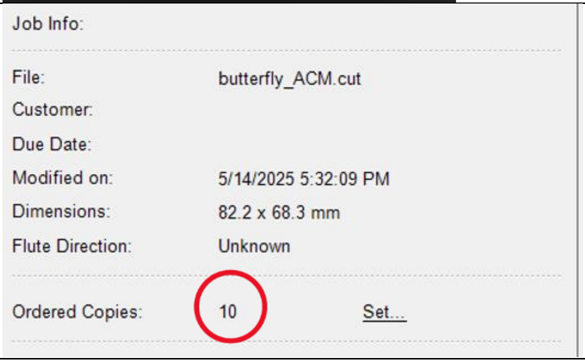
Tester	Magnus Trillhus Olsnes
Date	May 2025
Pass/Fail	Pass

Test 4 Set material/edit job

Requirements tested: R-2.3.1

Test ID: T-4

Steps	Test	Comment	Visual
1	Click on a job	Job details should appear on the right.	
2	Then click on a job and set the number of copies in the job details tab, click ok		 File name: ELEPHANTCOD.cgf Customer: Due date: Modified on: 2025-05-14 15:46 Dimensions: 420 X 297 mm Ordered Copies 1 Material: Paper 50g Cutting Profile: Tooling Preset: Optimization Preset: Kongsberg Default Production Preset: Mapping: Kongsberg Default

3	Check if the job details "Ordered copies" was updated to set number	Should update automatically, ordered copies has been set to 10 in this example.	 Set ordered copies Enter desired number of copies 10
4	Verify on IPC	The IPC should now be updated with the new data	 Job Info: File: butterfly_ACM.cut Customer: Due Date: Modified on: 5/14/2025 5:32:09 PM Dimensions: 82.2 x 68.3 mm Flute Direction: Unknown Ordered Copies: 10 Set...

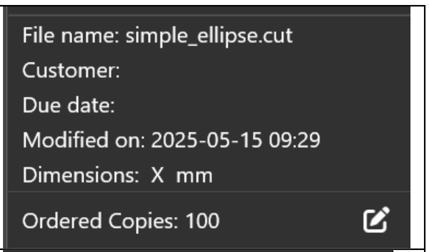
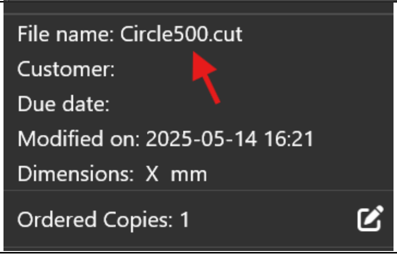
Tester	Magnus Trillhus Olsnes
Date	May 2025
Pass/Fail	Pass

Test 5 Job Data

Requirements tested: R-3.3.1, R-3.3.2

Test ID: T-5

Steps	Test	Comment	Visual
1	Click on a connected table	You should see a list of all active jobs on the table	

2	Click on one of the jobs	Job details should be on the right side of the page	 File name: simple_ellipse.cut Customer: Due date: Modified on: 2025-05-15 09:29 Dimensions: X mm Ordered Copies: 100
3	Click on another job	The job data should change to match the job you are on	 File name: Circle500.cut Customer: Due date: Modified on: 2025-05-14 16:21 Dimensions: X mm Ordered Copies: 1

Tester	Magnus Trillhus Olsnes
Date	May 2025
Pass/Fail	Pass

Test 6 Event handling

Requirements tested: R-3.2.1

Test ID: T-6

Steps	Test	Comment	Visual
1	1. Add a job to the queue from a table	Should update the KTT Application (Active jobs)	
2	2. Locate the table on the KTT app and click on it	Should show the new data	

Tester	Magnus Trillhus Olsnes
Date	May 2025
Pass/Fail	Pass

Test 7 Query for outdated information

Requirements tested: R-3.2.2

Test ID: T-7

Steps	Test	Comment	Visual

1	Disconnect the application		
2	Change some data on the HUB	Delete, add or edit some jobs in IPC for example	
3	Reconnect the application	The new data should be there	

Tester	Magnus Trillhus Olsnes
Date	May 2025
Pass/Fail	Pass

Test 8 Testing dashboard feature

Requirements tested: R-4.1.1

Test ID: T-8

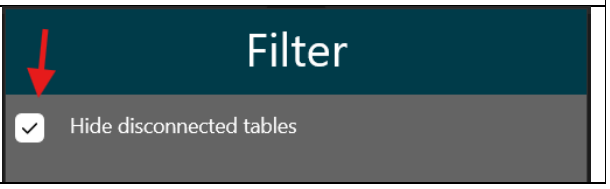
Steps	Test	Comment	Visual
1	Click on the dashboard button	Should be in the bottom right on the home page	↗
2	Exit dashboard, change the order of the tables with the drag and drop or use the filtering function		
3	Open the dashboard again	The dashboard should now be updated	

Tester	Magnus Trillhus Olsnes
Date	May 2025
Pass/Fail	Pass

Test 9 Data filtering

Requirements tested: R-4.1.3

Test ID: T-9

Steps	**Test**	**Comment**	**Visual**
1	Click on the filter button on the table overview (Home Page)	A Pop up window should appear in the middle of the screen	⏷
2	Check the 'Hide disconnected tables' box to filter out tables that are not connected	Should see it update right away	 Filter ☑ Hide disconnected tables

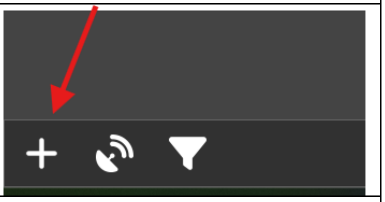
3	Click "outside" the pop up window to exit	All connected tables should now show	

Tester	Magnus Trillhus Olsnes
Date	May 2025
Pass/Fail	Pass

Test 10 Multiple tables

Requirements tested: R-5.1.1

Test ID: T-10

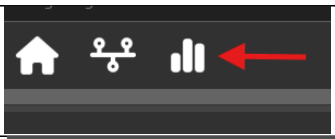
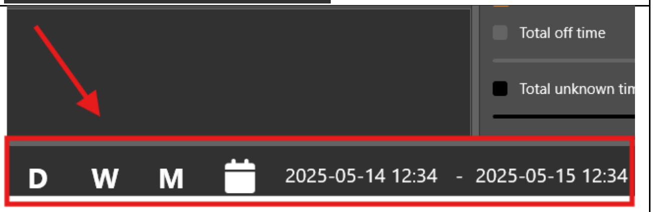
Steps	Test	Comment	Visual
1	Click the add table button in the application and enter the Credentials for a table		
2	Click Done	The table should be added to the table overview on the home page	
3	Do step 1-2 again	Should add more tables	

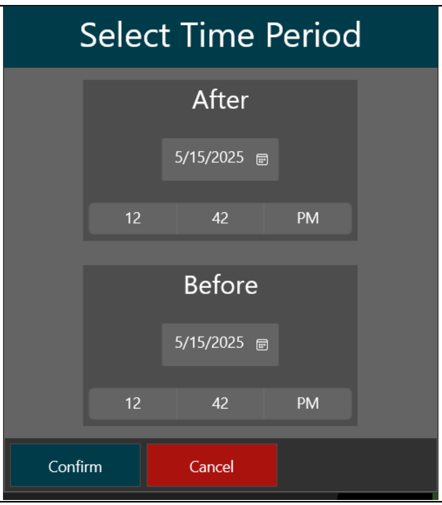
Tester	Magnus Trillhus Olsnes
Date	May 2025
Pass/Fail	Pass

Test 11 Displaying the time reports

Requirements tested: R-6.2.1

Test ID: T-11

Steps	Test	Comment	Visual
1	Navigate to the statistics page	Should open up the statistics page for all tables.	
2	Click on D(Day), W(Week), M(Month)	The app should show the time from the start date to the end date next to the buttons, they	 Total off time Total unknown tim D W M 2025-05-14 12:34 - 2025-05-15 12:34

		should match your input	
3	Click on the calendar and pick a start and end date	A pop up should appear in the middle of the screen	 Select Time Period After 5/15/2025 12 42 PM Before 5/15/2025 12 42 PM Confirm Cancel
4	Click confirm	Data should update for the given time	
5	Navigate to home page, click on a table and click on the statistics button and repeat steps 1-4	The same but for the job statistics	

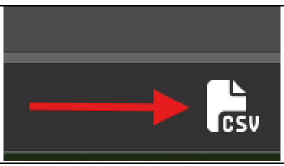
Tester	Magnus Trillhus Olsnes
Date	May 2025
Pass/Fail	Pass

Test 12 Job Report

Requirements tested: R-6.1.1

Test ID: T-12

Steps	Test	Comment	Visual
1	Click on a table and navigate to the table statistics page		

2	Locate the download CSV button in the bottom right and click download	File explorer should come up and you can save it on your computer	
3	Open the file	The job report should be displayed	

Tester	Magnus Trillhus Olsnes
Date	May 2025
Pass/Fail	Pass

Test 14 Testing GUI

- Precondition for this test: Tester is not part of the dev team

Requirements tested: R-3.3.4

Test ID: T-14

Steps	**Test**	**Comment**	**Visual**
1	Have the tester(s) try out the application. Testers will give their opinion if the GUI is easy to understand, navigate and pleasant to look at.		
2	Collect feedback from the testers	The app should be intuitive and easy to use for anyone	

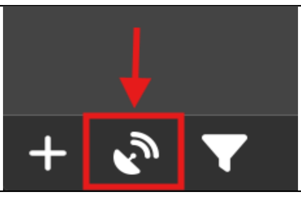
Testers	Ole Magnus (KPCS)
Date	May 2025
Feedback	**Feedback in the results section of this file**

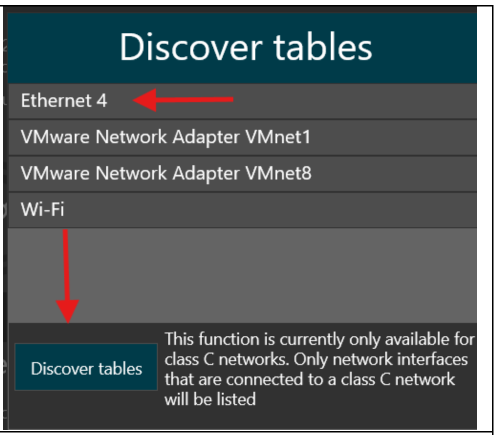
Test 15 Scan Service

- Preconditions: Be on a C class network and on the same network

Requirements tested: R-5.2.1

Test ID: T-15

Steps	**Test**	**Comment**	**Visual**
1	In the bottom left, click the scan function button	A pop up window should appear.	

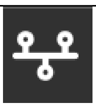
2	Select your current network connection		 Discover tables Ethernet 4 VMware Network Adapter VMnet1 VMware Network Adapter VMnet8 Wi-Fi Discover tables This function is currently only available for class C networks. Only network interfaces that are connected to a class C network will be listed
3	Click Discover tables	All tables on the network should be updated to the table overview on the home page	

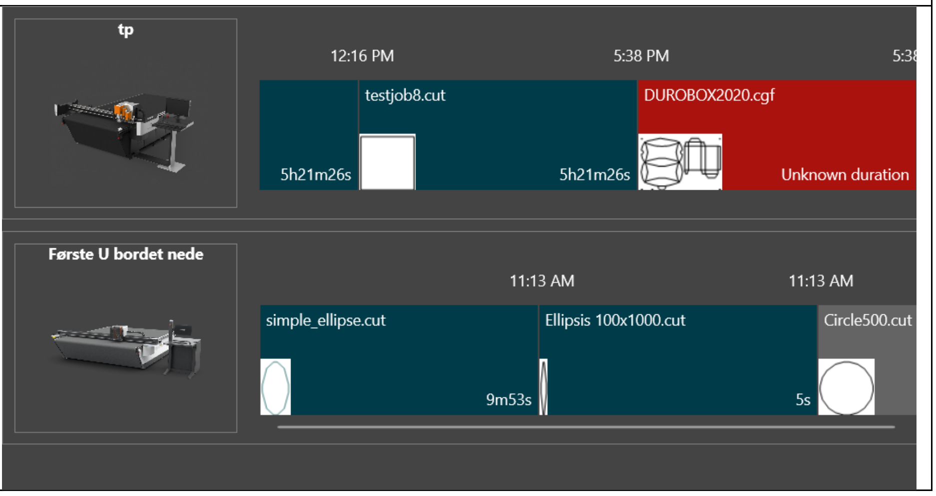
Tester	**Magnus Trillhus Olsnes**
Date	**May 2025**
Pass/Fail	**Pass**

Test 16 Timeline job display

Requirements tested: R-4.1.2

Test ID: T-16

Steps	Test	Comment	Visual
1	Navigate to the timeline	The timeline should contain a list of jobs for all the tables connected to the application	

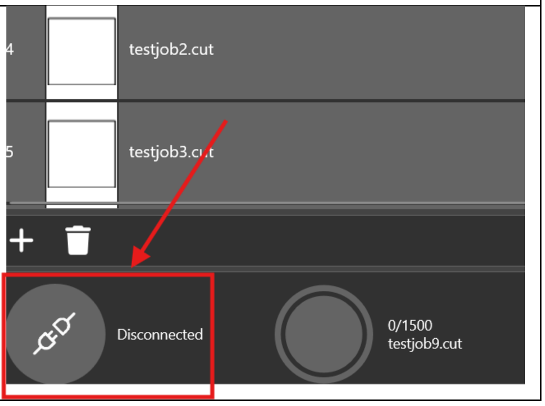
		with the different statuses of the jobs.	
2	Scroll to the right to view all the jobs of a table	Statuses of the tables should show	 tp 12:16 PM 5:38 PM 5:38 testjob8.cut DUROBOX2020.cgf 5h21m26s 5h21m26s Unknown duration Første U bordet nede 11:13 AM 11:13 AM simple_ellipse.cut Ellipsis 100x1000.cut Circle500.cut 9m53s 5s

Tester	Magnus Trillhus Olsnes
Date	May 2025
Pass/Fail	Pass

Test 17 operator required actions

Requirements tested: R-1.2.2

Test ID: T-17

Steps	Test	Comment	Visual
1	Click a table		
2	Locate status message	Her you should see the different statuses for the table in real time	 4 testjob2.cut 5 testjob3.cut Disconnected 0/1500 testjob9.cut

3	Go on IPC to update a table	The status should change in the application too	

Tester	Magnus Trillhus Olsnes
Date	May 2025
Pass/Fail	Pass

Test 18 Store table information

Requirements tested: R-5.1.3

Test ID: T-18

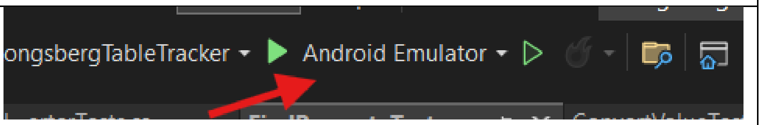
Steps	Test	Comment	Visual
1	Exit the application		
2	Open the application	Tables should still be there with the correct credentials, so you should not have to add or scan for them again	

Tester	Magnus Trillhus Olsnes
Date	May 2025
Pass/Fail	Pass

Test 13 Multiple platforms

Requirements tested: R-7.1.1

Test ID: T-13

Steps	**Test**	**Comment**	**Visual**
1	Go through the testing routine again, but with the application running on a smartphone next to the desktop application (same configuration)	The emulator should mirror the desktop version.	

Tester	Magnus Trillhus Olsnes
Date	May 2025
Pass/Fail	Fail

5. Test Results

Test ID	Comment
T-1	OK
T-2	OK
T-3	OK
T-4	OK
T-5	OK
T-6	OK
T-7	OK
T-8	OK
T-9	OK
T-10	OK
T-11	OK
T-12	OK
T-13	We have not had the time, but the app could in the future run on mobile devices. Since we developed the app in .NET MAUI, the support is there.
T-14	1. The pop up window when you choose a date could be a little more intuitive. 2. A way to tell where you are in the application, especially home page. 3. More tooltips around the application. Else it was very good
T-15	OK
T-16	At the moment we are not handling all the statuses, but still enough to pass our test
T-17	OK
T-18	OK

I TDD Model To ViewModel Messaging

Document Information

Field	Details
Title	Handling Communication Between a Self-Updating Model and the ViewModels
Author	Ole William Skistad Huslende
Date	14.4.2025
Status	Proposed / Approved / Rejected
Reviewed By	Tormod Smidesang

Summary

This document evaluates methods for real-time communication between a self-updating Model (using Server-Sent Events) and multiple ViewModels in an MVVM architecture. The options compared are:

- **IMessenger from CommunityToolkit.MVVM**
- **Event-based communication**
- **Direct push from Model to ViewModel**

The goal is to identify the most and MVVM friendly solution.

I.1 Context

What is the technical problem?

The Model layer receives updates via SSE and must notify ViewModels. The structure lacks MVVM implementaion

What is the goal?

- Ensure real-time updates
- Maintain MVVM separation of concerns
- Allow for testing, scalability, and maintainability
- Prevent memory leaks and threading issues

I.2 Options Considered

Option 1: IMessenger (MVVM Toolkit)

Overview: Use IMessenger to publish messages from the Model. ViewModels implement IRecipient to receive them.
Pros:

- Fully decoupled
- Built-in thread safety
- Scalable to many ViewModels
- Easy to mock and test
- Compatible with .NET MAUI

Cons:

- Slightly more boilerplate
- Requires clear message naming conventions

Option 2: Event-Based Communication

Overview: Models expose events which ViewModels subscribe to.
Pros:

- Real-time updates
- Preserves loose coupling
- Compatible with INotifyPropertyChanged

Cons:

- Manual threading control required
- Risk of memory leaks if unsubscribing is missed
- Complexity increases with multiple events

Option 3: Direct Push

Overview: The Model holds references to ViewModels and updates them directly.
Pros:

- Simple and immediate updates

Cons:

- Tight coupling violates MVVM principles
- Difficult to test and maintain
- Not scalable to multiple ViewModels

I.3 Decision

Chosen Option: Option 1 – IMessenger
Selected for clear MVVM structur and it integrates well with the existing .NET MAUI ecosystem.

Trade-offs

- Additional boilerplate
- Requires message naming conventions and structure

I.4 Impact

Changes Introduced

- Integrate IMessenger in the communication layer
- Refactor Models to send messages
- Update ViewModels to use IRecipient

Affected Components

- Model Layer
- ViewModel Layer

Timeline

- 1–2 days: Initial setup and message structure
- 1 week: Migration of all SSE-related features

I.5 Related Documents

- CommunityToolkit.MVVM – Messenger [57]
- .NET Events [58]
- MVVM structure [28]

J Application User Manual

User Manual for the KTT Application

Description

This manual will help you understand how to use the KTT application. The application is made for users who work with Kongsberg cutting tables. It shows you what the tables are doing and lets you upload and manage jobs on them.

The guide starts with an overview of the features on the different tables so the user can familiarize themselves with the buttons and features before diving into the steps for using the application.

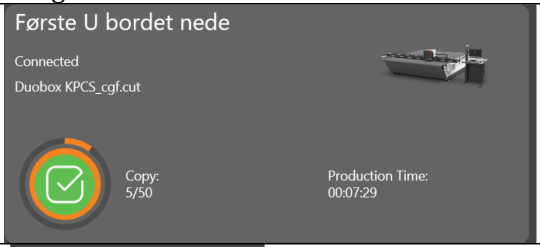
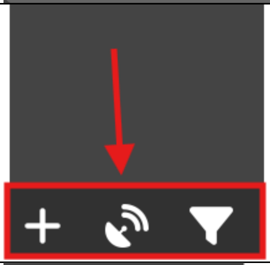
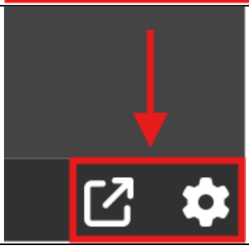
Then the guide starts from the beginning, when you first open the application. It explains to the user how to do different tasks step by step. You will learn how to add a table, scan for tables, upload jobs, and see statistics and history from the tables. There is also a part about changing the language in the settings.

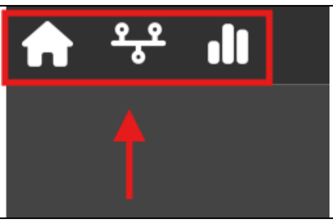
Each part of the manual gives you clear steps, explains what happens, and has space for pictures so it is easier to follow. This guide is made to be easy to read, even if you are new to the system.

Please have the application open and follow the steps along with the guide. Now let's start with the Page overviews to get a sniff of the features and the buttons.

1. Main Page Overview

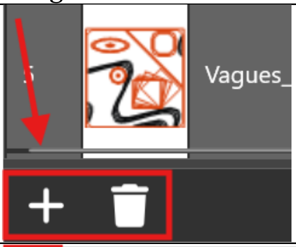
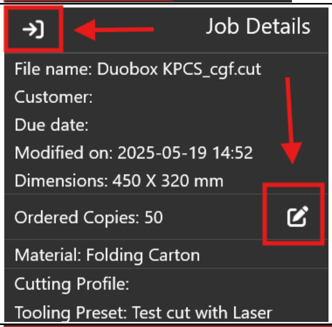
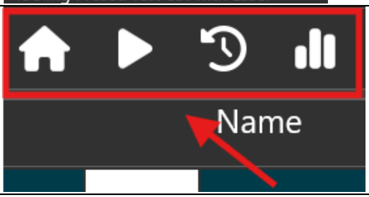
When you enter the application, you arrive at the Home Page. Below is an overview of the buttons and features available on this page:

Feature	Description	Image
Table Cards	Displays connected tables with name and status. Click for more details.	 Første U bordet nede Connected Duobox KPCS_cgf.cut Copy: 5/50 Production Time: 00:07:29
1.Add Table 2.Scan Button 3.Filter Button	1. Add a new table using IP, hostname, client ID and secret. 2. Discover tables on the same network. 3. Filter to hide disconnected tables.	
1.Dashboard Button 2. Settings Button	1. Open dashboard to view tables. 2. Access language selection.	

1.Home Button 2.Timeline Button 3. MetaStatistics for tables button	1.Return to mainpage 2. View a timeline of jobs across all tables. 3. View total production time for all tables.	

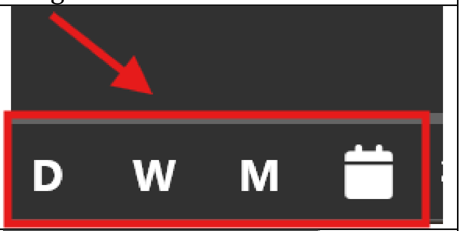
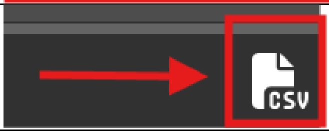
2. Table Overview

Once you enter a table new features arise, below is an overview of the buttons and features available on this page:

Feature	Description	Image
1. Add Job button 2. Delete Job button	1.Adds a job to the table 2.Deletes a job on the table	
1.Hide job details button 2. Set copies button	1.Removes the jobdetails on the screen until you click a job again 2.Manually set number of copies for a specific job	Job Details File name: Duobox KPCS_cgf.cut Customer: Due date: Modified on: 2025-05-19 14:52 Dimensions: 450 X 320 mm Ordered Copies: 50 Material: Folding Carton Cutting Profile: Tooling Preset: Test cut with Laser
1.Home button 2.ActiveJobs button 3.HistoricalJobs button 4.Statistics button	1.Return to MainPage 2.View the Active jobs 3.View the Historical jobs 4. View the statistics for the current table you are on.	

3.MetaStatisticsPage Overview

Once you click the statistics button inside a table (metastatistics), new features arise, below is an overview of the buttons and features available on this page:

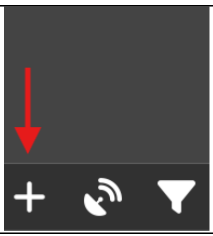
Feature	Descripiton	Image
1.Set day button 2.Set week button 3.Set month button 4.Custom Date button	1.View data from the last 24h 2.View data from the last week 3.View data from the last month 4.view data from set date	
1.Download button	Download a CSV file to view the data	

3.StatisticsPage Overview

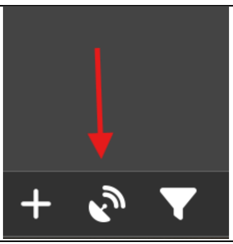
Once you click on the statistics, the same features exist as in 3 except for the download button.

Steps:

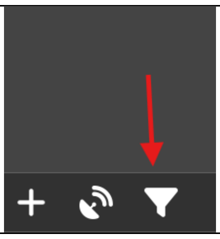
1. Adding a Table (Main Page)

Step	Description	Visual
1	Click on the add table button and write in the credentials given for the table. Your table should now be added to the application	

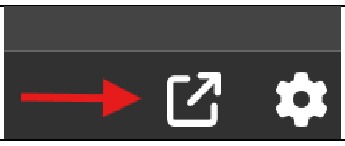
2. Scanning for Tables (Main Page)

Step	Description	Visual
1	Click on the Scan button and choose the network type, for example ethernet or wifi and then on “discover tables” Note! this runs in the background and may take a minute	

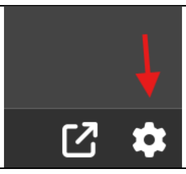
3.Filtering tables (Main Page)

Step	Description	Visual
1	Click on the filter button and check off “Hide disconnected tables” Now you should only see the connected tables	

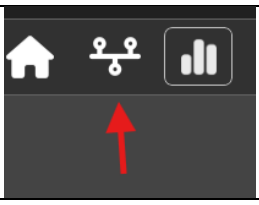
4.Dashboard function (Main Page)

Step	Description	Visual
1	Click on the dashboard button, the dashboard should come out as a separate page.	

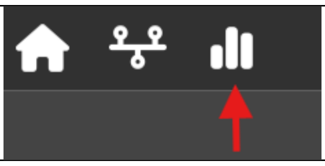
5.Settings (Main Page)

Step	Description	Visual
1	Click on the settings button, here you can select your language. Please restart the application after doing so.	

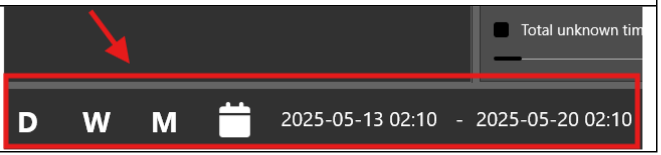
6.Timeline (Main Page)

Step	Description	Visual
1	Click on the timeline button to get the timeline of the tables connected with their current jobs.	

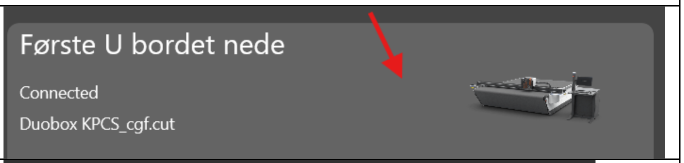
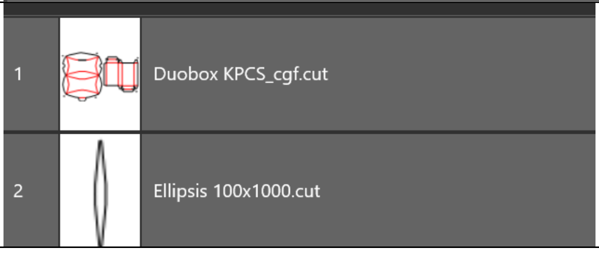
7.Table Statistics (Main Page)

Step	Description	Visual
1	Click on the Statistics for tables button, here you will get statistics data for the tables	

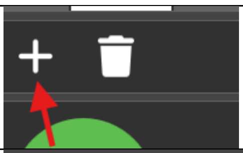
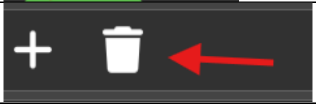
8.Table Statistics (Table statistics)

Step	Description	Visual
1	Once on this page, you have the option to change dates from where you get the data from, so you can retrieve data for the last day, week, month or choose a date yourself.	 Total unknown tim D W M 2025-05-13 02:10 - 2025-05-20 02:10

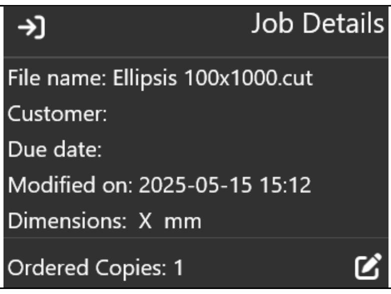
9.View jobs (Main Page)

Step	Description	Visual
1	On main page, you can enter a table by simply clicking on it.	
2	Once inside the table, you can view all the jobs, both historical and active.	

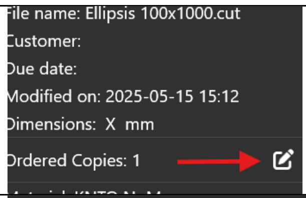
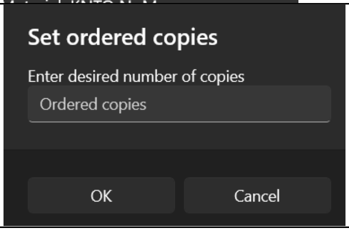
10.Adding or deleting jobs (Table)

Step	Description	Visual
1	Once inside the table click on the add job button to upload a job from your computer.	
2	Then select a job and click on the delete button, this will delete the selected job.	

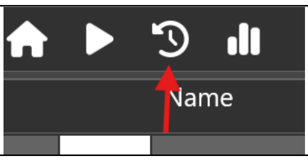
11.Job Details (Table)

Step	Description	Visual
1	When you enter a table, click on a job to get the job details for the selected job	 Job Details File name: Ellipsis 100x1000.cut Customer: Due date: Modified on: 2025-05-15 15:12 Dimensions: X mm Ordered Copies: 1

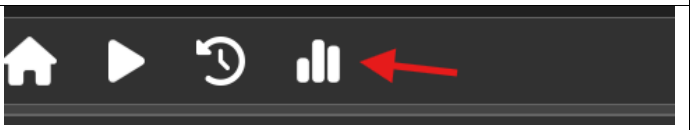
12.Set number of copies (Table)

Steps	Description	Visual
1	Once inside a table, on job details you can click the set copies button to select the number of copies you want for the job.	 ile name: Ellipsis 100x1000.cut ustomer: ue date: Modified on: 2025-05-15 15:12 imensions: X mm rdered Copies: 1
2	A pop up window will appear, and you can type in the number and simply press ok	 Set ordered copies Enter desired number of copies Ordered copies OK Cancel

13.Historical jobs (Table)

Step	Description	Visual
1	Click on the historical jobs button to view the list of historical jobs on the table you are currently on	

14.Statistics for a table (Table)

Step	Description	Visual
1	When on a table, select the statistics button and view all the data for the table you are currently on	

K Code Documentation

Kongsberg Table Tracker

Generated by Doxygen 1.9.1

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

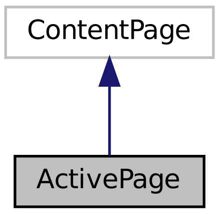
Chapter 3

Class Documentation

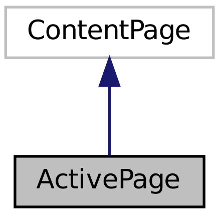
3.1 ActivePage Class Reference

Activepage is the page that shows the active jobs for a table.

Inheritance diagram for ActivePage:



Collaboration diagram for ActivePage:



Public Member Functions

- ActivePage (TableContextService tableContextService, InfoPanelContentView infoPanelContentView)

  The constructor for the Activepage.

Protected Member Functions

- override void OnAppearing ()

  Called when the page is appearing. Sets the binding context for the page and the info panel. Also resets the selected job in the active view model to null.

3.1.1 Detailed Description

Activepage is the page that shows the active jobs for a table.

Author

Ole William Skistad Huslende

Tormod Smidesang

Elvin Andreas Pedersen

3.1.2 Constructor & Destructor Documentation

3.1.2.1 ActivePage()

```
ActivePage.ActivePage (
            TableContextService tableContextService,
            InfoPanelContentView infoPanelContentView )  [inline]
```

The constructor for the Activepage.

Parameters

tableContextService	Dependency injection for the table context service.
infoPanelContentView	Dependency injection for the info panel content view.

3.1.3 Member Function Documentation

3.1.3.1 OnAppearing()

```
override void ActivePage.OnAppearing ( )  [inline], [protected]
```

Called when the page is appearing. Sets the binding context for the page and the info panel. Also resets the selected job in the active view model to null.

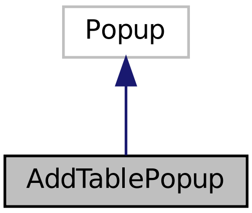
The documentation for this class was generated from the following file:

- Views/TableContextViews/ActivePage.xaml.cs

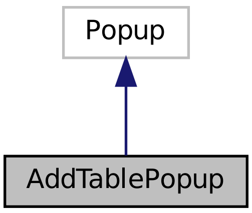
3.2 AddTablePopup Class Reference

Popup for adding a new table.

Inheritance diagram for AddTablePopup:



Collaboration diagram for AddTablePopup:



Public Member Functions

- AddTablePopup (AddTableViewModel addTableViewModel)

    Initializes a new instance of the AddTablePopup class.

Properties

- Command CloseCommand `[get]`

  Command to close the popup.

3.2.1 Detailed Description

Popup for adding a new table.

Author

Ole William Skistad Huslende

Tormod Smidesang

Elvin Andreas Pedersen

3.2.2 Constructor & Destructor Documentation

3.2.2.1 AddTablePopup()

```
AddTablePopup.AddTablePopup (
            AddTableViewModel addTableViewModel )  [inline]
```

Initializes a new instance of the AddTablePopup class.

Parameters

addTableViewModel	The view model for the add table popup.

3.2.3 Property Documentation

3.2.3.1 CloseCommand

```
Command AddTablePopup.CloseCommand  [get]
```

Command to close the popup.

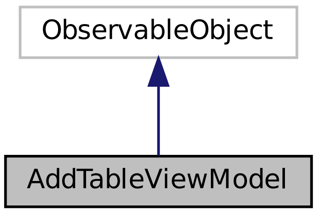
The documentation for this class was generated from the following file:

- Views/Popups/AddTablePopup.xaml.cs

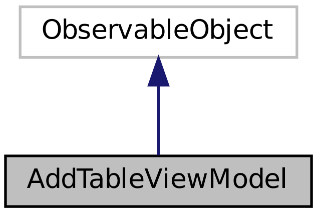
3.3 AddTableViewModel Class Reference

ViewModel for the AddTablePopup.

Inheritance diagram for AddTableViewModel:



Collaboration diagram for AddTableViewModel:



Public Member Functions

- AddTableViewModel (IServiceProvider serviceProvider, TableEntryFieldValidator tableEntryFieldValidator, TableContextService tableContextService)

  Constructor for the AddTableViewModel.
- void SetPopup (AddTablePopup Popup)

  Sets the popup for adding a table.

Properties

- string NameError `[get]`

  The error message for the table name field.
- string IpError `[get]`

  The error message for the IP address field.
- string ClientIdError `[get]`

The error message for the client ID field.
- string ClientSecretError `[get]`

  The error message for the client secret field.
- bool IsFormValid `[get]`

  Checks if the form is valid.

3.3.1 Detailed Description

ViewModel for the AddTablePopup.

AddTablePopup is a popup that allows the user to add a new table to the application. Showing when you press the add button in the mainpage.

Author

Ole William Skistad Huslende

Tormod Smidesang

Elvin Andreas Pedersen

3.3.2 Constructor & Destructor Documentation

3.3.2.1 AddTableViewModel()

```
AddTableViewModel.AddTableViewModel (
            IServiceProvider serviceProvider,
            TableEntryFieldValidator tableEntryFieldValidator,
            TableContextService tableContextService )  [inline]
```

Constructor for the AddTableViewModel.

Parameters

serviceProvider	Service provider for dependency injection.
tableEntryFieldValidator	Validator for the table entry fields.
tableContextService	Service for managing table contexts.

3.3.3 Member Function Documentation

3.3.3.1 SetPopup()

```
void AddTableViewModel.SetPopup (
            AddTablePopup Popup )  [inline]
```

Sets the popup for adding a table.

Parameters

Popup	The popup

3.3.4 Property Documentation

3.3.4.1 ClientIdError

```
string AddTableViewModel.ClientIdError  [get]
```

The error message for the client ID field.

3.3.4.2 ClientSecretError

```
string AddTableViewModel.ClientSecretError  [get]
```

The error message for the client secret field.

3.3.4.3 IpError

```
string AddTableViewModel.IpError  [get]
```

The error message for the IP address field.

3.3.4.4 IsFormValid

```
bool AddTableViewModel.IsFormValid  [get]
```

Checks if the form is valid.

3.3.4.5 NameError

```
string AddTableViewModel.NameError  [get]
```

The error message for the table name field.

The documentation for this class was generated from the following file:

- ViewModels/Popups/AddTableViewModel.cs

3.4 ApiService Class Reference

Service for handling communication with the HUB

Public Member Functions

- ApiService (IMessenger messenger)

  Constructor.
- string GetEvent ()

  Returns the oldest unhandled event from the queue
- async void SetIpAddress (IPAddress? newIpAddress)

  Sets a new IP address, ensures the event listener for the previous IP address is stopped. If IP is null, sets badIp↵ Address to true in order to prevent connection attempts from being made.
- async Task CredentialsUpdated (string? ClientId, string? ClientSecret)

  Handles changes to API credentials. Stops event listener, calls EnsureValidAuthentication(string?, string?)
- async Task< bool > EnsureValidAuthentication (string? ClientId, string? ClientSecret)

  Ensures the provided credentials are valid and generates a new access token if one does not already exist or an existing token is expired.
- bool IsEventListenerRunning ()

  Checks if the event listener is running
- async void StartEventListener ()

  Starts the event listener unless it is already running, in which case nothing will happen.
- async Task StopEventListener ()

  Stops the even listener unless it is already stopped, in which case nothing will happen
- async Task EventListener (CancellationToken token)

  Listens to events until cancelled. Events are pushed to eventBuffer Will attempt to reconnect every 5 seconds when connection problems arise.
- string? ExtractJsonFromPrefixedMessage (string message)

  Gets the start of the JSON string. Used by the event listener
- async Task< TableAuthentication?> RequestAccessToken (string ClientId, string ClientSecret)

  Requests an access token using the provided credentials
- async Task< string?> GetHello ()

  Attempts to get a response from the /hello endpoint. Used to verify that there is a cutting table on the provided IP address
- async Task< JobQueue?> GetActiveJobs ()

  Gets the active job queue from the table
- async Task< JobQueue?> GetHistoryJobs ()

  Gets the history job queue from the table
- async Task< JobTask?> GetHistoryJob (string jobId)

  Gets a specific job from the history queue.
- async Task< ImageSource?> GetImageOfJob (string jobId, string value)

  Gets a preview image of a job
- async Task DeleteJobTask (string jobId)

  Delete the specified job from the table
- async Task SetOrderedCopies (string jobId, int Amount)

  Set ordered copies for the specified job
- async Task< ImageSource?> GetImageOfTable ()

  Gets an image of the table
- async Task< List< TableProperty >?> GetAllTableProperties ()

  Gets all the system properties of the table

- async Task< List< QueueServerJobStatistics?> > GetProductionStatistics (DateTime before, DateTime after)

  Gets production statistics within the given timespan
- async Task< List< PropertyTrends?> > GetPropertyTrends (DateTime before, DateTime after, string property)

  Gets property trends from the table
- async Task< TResponse?> PostAsync< TRequest, TResponse > (string url, TRequest requestBody)

  Generic function to handle HTTP post requests that are expected to return an object serialized to JSON
- async Task PutAsync< TRequest > (string url, TRequest requestBody)

  Generic function to handle HTTP put requests
- async Task PostActiveJob (Stream requestBody)

  Sends a job in zip format to the table
- async Task< TResponse?> GetAsync< TResponse > (string url)

  Generic function to handle HTTP get requests that are expected to return an object serialized to JSON
- async Task DeleteAsync (string url)

  Generic function to handle HTTP delete requests

Public Attributes

- readonly JsonSerializerOptions jsonSerializerOptions

  An object containing options used for deserializing data received from the HUB in JSON format

3.4.1 Detailed Description

Service for handling communication with the HUB

Author

Ole William Skistad Huslende

Tormod Smidesang

3.4.2 Constructor & Destructor Documentation

3.4.2.1 ApiService()

```
ApiService.ApiService (
            IMessenger messenger )  [inline]
```

Constructor.

Parameters

messenger	The table context messenger

3.4.3 Member Function Documentation

3.4.3.1 CredentialsUpdated()

```
async Task ApiService.CredentialsUpdated (
             string?  ClientId,
             string?  ClientSecret )  [inline]
```

Handles changes to API credentials. Stops event listener, calls EnsureValidAuthentication(string?, string?)

Parameters

ClientId	The new client ID
ClientSecret	The new client secret

Returns

Task representing an asynchronous operation

3.4.3.2 DeleteAsync()

```
async Task ApiService.DeleteAsync (
             string url )  [inline]
```

Generic function to handle HTTP delete requests

Parameters

url	The endpoint to send the request to

Returns

A task representing an asynchronous operation

Exceptions

HttpRequestException	Thrown if the server returns an error code (400)

3.4.3.3 DeleteJobTask()

```
async Task ApiService.DeleteJobTask (
             string jobId )  [inline]
```

Delete the specified job from the table

Parameters

job↩ Id	The Id of the job to delete

Returns

Task representing an asynchronous operation

3.4.3.4 EnsureValidAuthentication()

```
async Task<bool> ApiService.EnsureValidAuthentication (
             string?  ClientId,
             string?  ClientSecret )  [inline]
```

Ensures the provided credentials are valid and generates a new access token if one does not already exist or an existing token is expired.

Parameters

ClientId	The client ID
ClientSecret	The client secret

Returns

True if the provided credentials are valid, false otherwise

3.4.3.5 EventListener()

```
async Task ApiService.EventListener (
             CancellationToken token )  [inline]
```

Listens to events until cancelled. Events are pushed to eventBuffer Will attempt to reconnect every 5 seconds when connection problems arise.

Parameters

token	Used to cancel the event listener

Returns

Task representing an asynchronous operation

3.4.3.6 ExtractJsonFromPrefixedMessage()

```
string?  ApiService.ExtractJsonFromPrefixedMessage (
             string message )  [inline]
```

Gets the start of the JSON string. Used by the event listener

Parameters

message	The json to extract

Returns

The extracted json, or null if something goes wrong

3.4.3.7 GetActiveJobs()

```
async Task<JobQueue?> ApiService.GetActiveJobs ( )  [inline]
```

Gets the active job queue from the table

Returns

A JobQueue object if successful, null otherwise

3.4.3.8 GetAllTableProperties()

```
async Task<List<TableProperty>?> ApiService.GetAllTableProperties ( )  [inline]
```

Gets all the system properties of the table

Returns

A list of TableProperty objects if successful, default otherwise

3.4.3.9 GetAsync< TResponse >()

```
async Task<TResponse?> ApiService.GetAsync< TResponse > (
             string url )  [inline]
```

Generic function to handle HTTP get requests that are expected to return an object serialized to JSON

Template Parameters

TResponse	The type of object expected to be returned by the endpoint specified

Parameters

url	The endpoint to send the get request to

Returns

If successful, returns the response to the request, deserialized to an object of the specified type. Otherwise, returns default

3.4.3.10 GetEvent()

```
string ApiService.GetEvent ( )  [inline]
```

Returns the oldest unhandled event from the queue

Returns

An event in JSON format, or an empty string if the queue is empty

3.4.3.11 GetHello()

```
async Task<string?> ApiService.GetHello ( )  [inline]
```

Attempts to get a response from the /hello endpoint. Used to verify that there is a cutting table on the provided IP address

Returns

The response from the endpoint, or null if no response could be retrieved

3.4.3.12 GetHistoryJob()

```
async Task<JobTask?> ApiService.GetHistoryJob (
             string jobId )  [inline]
```

Gets a specific job from the history queue.

Parameters

job↩ Id	The Id of the job to retrieve from the table

Returns

A JobTask object if successful, null otherwise

3.4.3.13 GetHistoryJobs()

```
async Task<JobQueue?> ApiService.GetHistoryJobs ( )  [inline]
```

Gets the history job queue from the table

Returns

A JobQueue object if successful, null otherwise

3.4.3.14 GetImageOfJob()

```
async Task<ImageSource?> ApiService.GetImageOfJob (
             string jobId,
             string value )  [inline]
```

Gets a preview image of a job

Parameters

jobId	The Id of the job to request a preview image of
value	The type of image to request

Returns

An ImageSource object if successful, default otherwise

3.4.3.15 GetImageOfTable()

```
async Task<ImageSource?> ApiService.GetImageOfTable ( )  [inline]
```

Gets an image of the table

Returns

An ImageSource object depicting the table if successful, or null otherwise

3.4.3.16 GetProductionStatistics()

```
async Task<List<QueueServerJobStatistics?> > ApiService.GetProductionStatistics (
            DateTime before,
            DateTime after )  [inline]
```

Gets production statistics within the given timespan

Parameters

before	Get statistics before this point in time
after	Get statistics after this point in time

Returns

A list of QueueServerJobStatistics objects if successful, or an empty list otherwise

3.4.3.17 GetPropertyTrends()

```
async Task<List<PropertyTrends?> > ApiService.GetPropertyTrends (
            DateTime before,
            DateTime after,
            string property )  [inline]
```

Gets property trends from the table

Parameters

before	Get trends before this point in time
after	Get trends after this point in time
property	The property to get trends for

Returns

A list of PropertyTrends objects if successful, or an empty list otherwise

3.4.3.18 IsEventListenerRunning()

```
bool ApiService.IsEventListenerRunning ( )  [inline]
```

Checks if the event listener is running

Returns

True if the event listener is running, false otherwise

3.4.3.19 PostActiveJob()

```
async Task ApiService.PostActiveJob (
             Stream requestBody )  [inline]
```

Sends a job in zip format to the table

Parameters

requestBody	The request body

Returns

A task representing an asynchronous operation

Exceptions

UnauthorizedAccessException	Thrown if attempting to post a job when credentials are invalid or missing

3.4.3.20 PostAsync< TRequest, TResponse >()

```
async Task<TResponse?> ApiService.PostAsync< TRequest, TResponse > (
             string url,
             TRequest requestBody )  [inline]
```

Generic function to handle HTTP post requests that are expected to return an object serialized to JSON

Template Parameters

TRequest	The type of object for the request body
TResponse	The type of object expected to be returned by the endpoint specified

Parameters

url	The endpoint to send the post request to
requestBody	The request body

Returns

If successful, returns the response to the request, deserialized to an object of the specified type. Otherwise, returns default

Exceptions

UnauthorizedAccessException	Thrown if attempting to connect to a protected endpoint when credentials are invalid or missing

3.4.3.21 PutAsync< TRequest >()

```
async Task ApiService.PutAsync< TRequest > (
            string url,
            TRequest requestBody )  [inline]
```

Generic function to handle HTTP put requests

Template Parameters

TRequest	The type of object for the request body

Parameters

url	The endpoint to send the put request to
requestBody	The body of the request

Returns

A task object representing an asynchronous operation

Exceptions

HttpRequestException	Thrown if the HTTP response was anything but OK (200)

3.4.3.22 RequestAccessToken()

```
async Task<TableAuthentication?> ApiService.RequestAccessToken (
            string ClientId,
            string ClientSecret )  [inline]
```

Requests an access token using the provided credentials

Parameters

ClientId	The client ID
ClientSecret	The client secret

Returns

A TableAuthentication object containing the access token, or null if something goes wrong

3.4.3.23 SetIpAddress()

```
async void ApiService.SetIpAddress (
             IPAddress? newIpAddress )  [inline]
```

Sets a new IP address, ensures the event listener for the previous IP address is stopped. If IP is null, sets badIp↩ Address to true in order to prevent connection attempts from being made.

Parameters

newIpAddress	The new IP address

3.4.3.24 SetOrderedCopies()

```
async Task ApiService.SetOrderedCopies (
             string jobId,
             int Amount )  [inline]
```

Set ordered copies for the specified job

Parameters

jobId	The Id of the job for which to specify ordered copies
Amount	New value for ordered copies

Returns

Task representing an asynchronous operation

3.4.3.25 StartEventListener()

```
async void ApiService.StartEventListener ( )  [inline]
```

Starts the event listener unless it is already running, in which case nothing will happen.

3.4.3.26 StopEventListener()

```
async Task ApiService.StopEventListener ( )  [inline]
```

Stops the even listener unless it is already stopped, in which case nothing will happen

Returns

Task representing an asynchronous operation

3.4.4 Member Data Documentation

3.4.4.1 jsonSerializerOptions

```
readonly JsonSerializerOptions ApiService.jsonSerializerOptions
```

Initial value:

```
= new()
    {
        PropertyNameCaseInsensitive = true,
        Converters =
        {
            new EnumMemberJsonConverter<DataType>(),
            new JsonStringEnumConverter()
        }
    }
```

An object containing options used for deserializing data received from the HUB in JSON format

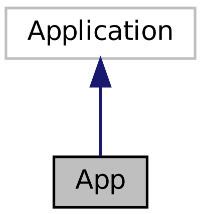
The documentation for this class was generated from the following file:

- Services/ApiService.cs

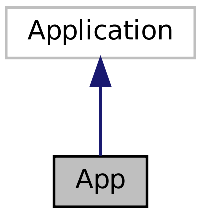
3.5 App Class Reference

This class is the entry point of the application.

Inheritance diagram for App:



Collaboration diagram for App:



Public Member Functions

- App (UserPreferences userPreferences)

  This constructor initializes the application and loads user preferences.

Protected Member Functions

- override Window CreateWindow (IActivationState? activationState)

  Creates the main window of the application.

3.5.1 Detailed Description

This class is the entry point of the application.

Author

Ole William Skistad Huslende

Tormod Smidesang

Elvin Andreas Pedersen

3.5.2 Constructor & Destructor Documentation

3.5.2.1 App()

```
App.App (
                UserPreferences userPreferences )  [inline]
```

This constructor initializes the application and loads user preferences.

Parameters

userPreferences	

3.5.3 Member Function Documentation

3.5.3.1 CreateWindow()

```
override Window App.CreateWindow (
            IActivationState?  activationState )  [inline], [protected]
```

Creates the main window of the application.

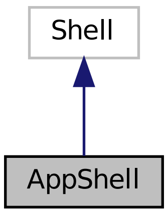
The documentation for this class was generated from the following file:

- App.xaml.cs

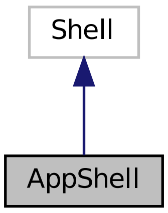
3.6 AppShell Class Reference

Appshell is the main shell of the application it contains the navigation structure.

Inheritance diagram for AppShell:



Collaboration diagram for AppShell:



3.6.1 Detailed Description

Appshell is the main shell of the application it contains the navigation structure.

Author

Ole William Skistad Huslende

Tormod Smidesang

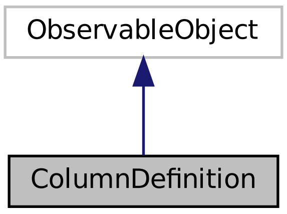
The documentation for this class was generated from the following file:

- AppShell.xaml.cs

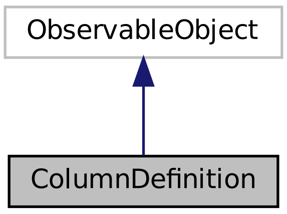
3.7 ColumnDefinition Class Reference

Represents a column definition for a table.

Inheritance diagram for ColumnDefinition:



Collaboration diagram for ColumnDefinition:



3.7.1 Detailed Description

Represents a column definition for a table.

Author

Elvin Andreas Pedersen

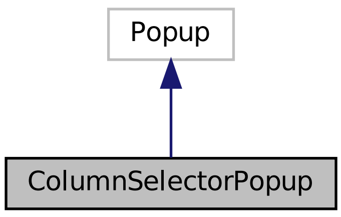
The documentation for this class was generated from the following file:

- Utilities/ColumnDefinition.cs

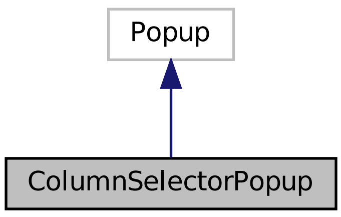
3.8 ColumnSelectorPopup Class Reference

Popup for selecting columns to display.

Inheritance diagram for ColumnSelectorPopup:



Collaboration diagram for ColumnSelectorPopup:



Public Member Functions

- ColumnSelectorPopup (ObservableCollection< ColumnDefinition > columnDefinitions)

  Initializes a new instance of the ColumnSelectorPopup class.

Properties

- ObservableCollection< ColumnDefinition > ColumnDefinitions `[get]`

  The collection of column definitions to be displayed in the popup.
- Command ClosePopupCommand `[get]`

  Command to close the popup.

3.8.1 Detailed Description

Popup for selecting columns to display.

Author

Elvin Andreas Pedersen

3.8.2 Constructor & Destructor Documentation

3.8.2.1 ColumnSelectorPopup()

```
ColumnSelectorPopup.ColumnSelectorPopup (
             ObservableCollection< ColumnDefinition > columnDefinitions )  [inline]
```

Initializes a new instance of the ColumnSelectorPopup class.

Parameters

columnDefinitions	Dependency injection of the column definitions to be displayed in the popup.

3.8.3 Property Documentation

3.8.3.1 ClosePopupCommand

```
Command ColumnSelectorPopup.ClosePopupCommand  [get]
```

Command to close the popup.

3.8.3.2 ColumnDefinitions

```
ObservableCollection<ColumnDefinition> ColumnSelectorPopup.ColumnDefinitions  [get]
```

The collection of column definitions to be displayed in the popup.

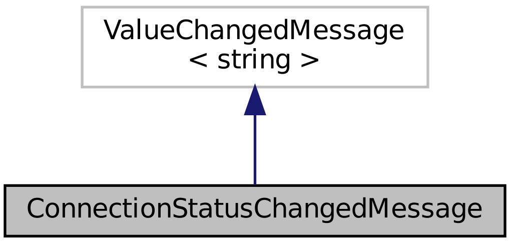
The documentation for this class was generated from the following file:

- Views/Popups/ColumnSelectorPopup.xaml.cs

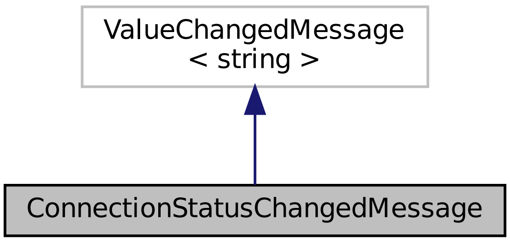
3.9 ConnectionStatusChangedMessage Class Reference

Message to send connection status changed to the viewmodels.

Inheritance diagram for ConnectionStatusChangedMessage:



Collaboration diagram for ConnectionStatusChangedMessage:



Public Member Functions

- **ConnectionStatusChangedMessage** (string value)

3.9.1 Detailed Description

Message to send connection status changed to the viewmodels.

This class is used to send messages between the model and the viewmodel with the help of IMessenger.

Author

Tormod Smidesang

The documentation for this class was generated from the following file:

- Models/ModelMessages.cs

3.10 CopyTime Class Reference

Represents a data model for the time used for each copy.

Properties

- DateTime? CopyStarted `[get, set]`

  The time when the copy operation started.
- DateTime? CopyEnded `[get, set]`

  The time when the copy operation ended.

3.10.1 Detailed Description

Represents a data model for the time used for each copy.

Author

Ole William Skistad Huslende

3.10.2 Property Documentation

3.10.2.1 CopyEnded

```
DateTime? CopyTime.CopyEnded [get], [set]
```

The time when the copy operation ended.

3.10.2.2 CopyStarted

```
DateTime? CopyTime.CopyStarted [get], [set]
```

The time when the copy operation started.

The documentation for this class was generated from the following file:

- Models/ApiStatistics/CopyTime.cs

3.11 CurrentStatistics Class Reference

Message sent to the statistics viewmodel to update the current statistics.

Properties

- DateTime **Before** `[get, set]`
- DateTime **After** `[get, set]`

3.11.1 Detailed Description

Message sent to the statistics viewmodel to update the current statistics.

Author

Ole William Skistad Huslende

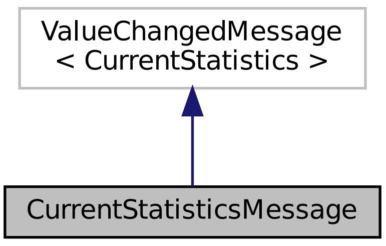
The documentation for this class was generated from the following file:

- ViewModels/ViewModelMessages.cs

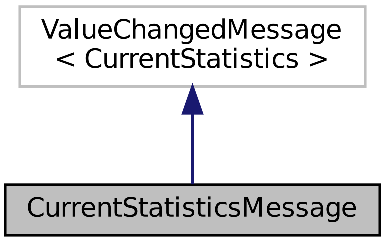
3.12 CurrentStatisticsMessage Class Reference

Sent the current statistics period from the metastatistics viewmodel to the statistics viewmodel.

Inheritance diagram for CurrentStatisticsMessage:



Collaboration diagram for CurrentStatisticsMessage:



Public Member Functions

- **CurrentStatisticsMessage** (CurrentStatistics value)

3.12.1 Detailed Description

Sent the current statistics period from the metastatistics viewmodel to the statistics viewmodel.

Author

Ole William Skistad Huslende

The documentation for this class was generated from the following file:

- ViewModels/ViewModelMessages.cs

3.13 Customer Class Reference

Properties

- string? CustomerId `[get, set]`

  Gets or sets the Id for the customer produced for.
- string? Name `[get, set]`

  Gets or sets the full name of the customer produced for.
- string? Country `[get, set]`

  Gets or sets the country where the customer produced for resides.
- string? City `[get, set]`

  Gets or sets the city where the customer produced for lives.
- string? PostalCode `[get, set]`

  Gets or sets the postal code for the customer produced for.
- string? Address `[get, set]`

  Gets or sets the address of the customer produced for.

3.13.1 Property Documentation

3.13.1.1 Address

```
string?  Customer.Address  [get], [set]
```

Gets or sets the address of the customer produced for.

3.13.1.2 City

```
string?  Customer.City  [get], [set]
```

Gets or sets the city where the customer produced for lives.

3.13.1.3 Country

```
string?  Customer.Country  [get], [set]
```

Gets or sets the country where the customer produced for resides.

3.13.1.4 CustomerId

```
string?  Customer.CustomerId  [get], [set]
```

Gets or sets the Id for the customer produced for.

3.13.1.5 Name

```
string?  Customer.Name  [get], [set]
```

Gets or sets the full name of the customer produced for.

3.13.1.6 PostalCode

```
string? Customer.PostalCode [get], [set]
```

Gets or sets the postal code for the customer produced for.

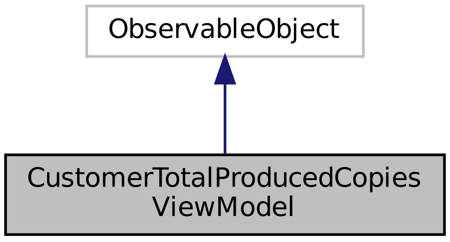
The documentation for this class was generated from the following file:

- Models/ApiQueue/Customer.cs

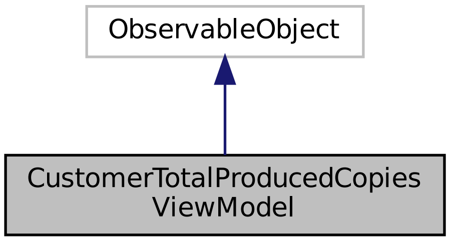
3.14 CustomerTotalProducedCopiesViewModel Class Reference

A view model for the customer total produced copies.

Inheritance diagram for CustomerTotalProducedCopiesViewModel:



Collaboration diagram for CustomerTotalProducedCopiesViewModel:



3.14.1 Detailed Description

A view model for the customer total produced copies.

Displays the customer and the total number of produced copies. Used in the statistics viewmodel.

Author

Ole William Skistad Huslende

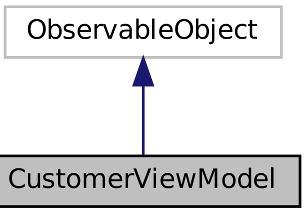
The documentation for this class was generated from the following file:

- ViewModels/TableContextViewModels/SubViewModels/CustomerTotalProducedCopiesViewModel.cs

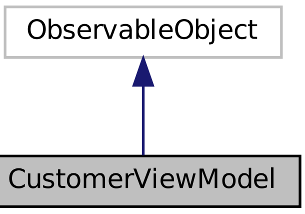
3.15 CustomerViewModel Class Reference

ViewModel for the customer.

Inheritance diagram for CustomerViewModel:



Collaboration diagram for CustomerViewModel:



Public Member Functions

- CustomerViewModel (IMessenger messenger, Customer customer)

  Constructor for the customer view model.
- override bool Equals (object? obj)

  Checks if two customer view models are equal. Used for comparing the customer view models in the list.
- override int GetHashCode ()

  Calculates the hash code for the customer view model. Used for comparing the customer view models in the list.

3.15.1 Detailed Description

ViewModel for the customer.

Purpose of this class is to provide only the interesting properties of the Customer.

Author

Ole William Skistad Huslende

3.15.2 Constructor & Destructor Documentation

3.15.2.1 CustomerViewModel()

```
CustomerViewModel.CustomerViewModel (
             IMessenger messenger,
             Customer customer )  [inline]
```

Constructor for the customer view model.

Parameters

messenger	Messenger used to send messages inside the table context.
customer	The customer to create the view model from.

3.15.3 Member Function Documentation

3.15.3.1 Equals()

```
override bool CustomerViewModel.Equals (
             object?  obj )  [inline]
```

Checks if two customer view models are equal. Used for comparing the customer view models in the list.

This method is used to compare the customer view models in the list. It compares the name, city, country and address of the customer view model.

Parameters

obj	Customer viewmodel

Returns

Returns true if customer view models are equal, else false

3.15.3.2 GetHashCode()

```
override int CustomerViewModel.GetHashCode ( )  [inline]
```

Calculates the hash code for the customer view model. Used for comparing the customer view models in the list.

This method is used to calculate the hash code for the customer view model. It calculates the hash code based on the name, city, country and address of the customer view model.

Returns

The hashcode

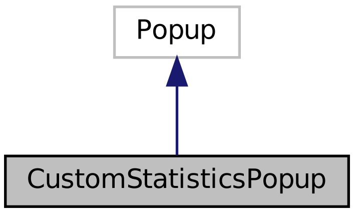
The documentation for this class was generated from the following file:

- ViewModels/TableContextViewModels/SubViewModels/CustomerViewModel.cs

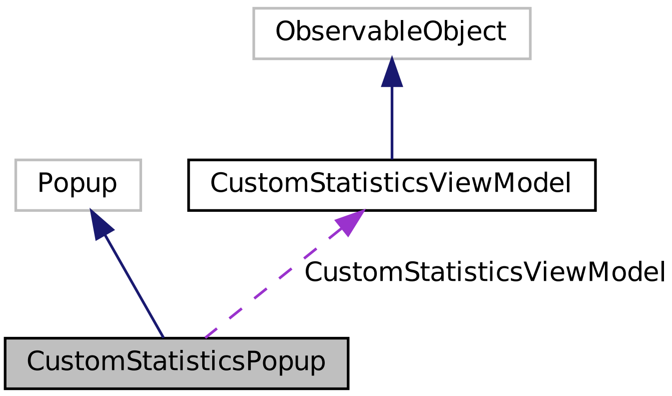
3.16 CustomStatisticsPopup Class Reference

Popup for displaying custom statistics.

Inheritance diagram for CustomStatisticsPopup:



Collaboration diagram for CustomStatisticsPopup:



Public Member Functions

- CustomStatisticsPopup (CustomStatisticsViewModel customStatisticsViewModel)

  Initializes a new instance of the CustomStatisticsPopup class.

Public Attributes

- CustomStatisticsViewModel CustomStatisticsViewModel

  The view model for the custom statistics popup.

Properties

- Command CloseCommand `[get]`

  Command to close the popup.

3.16.1 Detailed Description

Popup for displaying custom statistics.

Author

Ole William Skistad Huslende

Elvin Andreas Pedersen

3.16.2 Constructor & Destructor Documentation

3.16.2.1 CustomStatisticsPopup()

```
CustomStatisticsPopup.CustomStatisticsPopup (
            CustomStatisticsViewModel customStatisticsViewModel )  [inline]
```

Initializes a new instance of the CustomStatisticsPopup class.

Parameters

customStatisticsViewModel	Dependency injection of the view model for the custom statistics popup.

3.16.3 Member Data Documentation

3.16.3.1 CustomStatisticsViewModel

```
CustomStatisticsViewModel CustomStatisticsPopup.CustomStatisticsViewModel
```

The view model for the custom statistics popup.

3.16.4 Property Documentation

3.16.4.1 CloseCommand

```
Command CustomStatisticsPopup.CloseCommand  [get]
```

Command to close the popup.

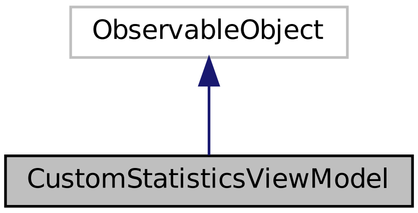
The documentation for this class was generated from the following file:

- Views/Popups/CustomStatisticsPopup.xaml.cs

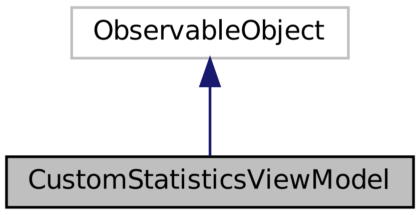
3.17 CustomStatisticsViewModel Class Reference

ViewModel for the CustomStatisticsPopup.

Inheritance diagram for CustomStatisticsViewModel:



Collaboration diagram for CustomStatisticsViewModel:



Public Member Functions

- CustomStatisticsViewModel (TableContextService tableContextService, IMessenger metaMessenger)

  Initializes a new instance of the CustomStatisticsViewModel class.
- void SetCaller (object caller)

  Sets the caller of the popup. This is used to determine which viewmodel to call the statistics request on.
- void SetPopup (CustomStatisticsPopup Popup)

  Sets the popup for custom statistics. This is used to close the popup when the user clicks the confirm or cancel button.

3.17.1 Detailed Description

ViewModel for the CustomStatisticsPopup.

CustomStatisticsPopup is used to display a popup for selecting a time range for production statistics.

Author

Ole William Skistad Huslende

3.17.2 Constructor & Destructor Documentation

3.17.2.1 CustomStatisticsViewModel()

```
CustomStatisticsViewModel.CustomStatisticsViewModel (
            TableContextService tableContextService,
            IMessenger metaMessenger )  [inline]
```

Initializes a new instance of the CustomStatisticsViewModel class.

Parameters

tableContextService	The table context service used to get the active table.
metaMessenger	The messenger used to send messages between the tableContext viewmodels and the external viewmodels.

3.17.3 Member Function Documentation

3.17.3.1 SetCaller()

```
void CustomStatisticsViewModel.SetCaller (
            object caller )  [inline]
```

Sets the caller of the popup. This is used to determine which viewmodel to call the statistics request on.

Parameters

caller	

3.17.3.2 SetPopup()

```
void CustomStatisticsViewModel.SetPopup (
            CustomStatisticsPopup Popup )  [inline]
```

Sets the popup for custom statistics. This is used to close the popup when the user clicks the confirm or cancel button.

Parameters

Popup	The CustomStatistics Popup

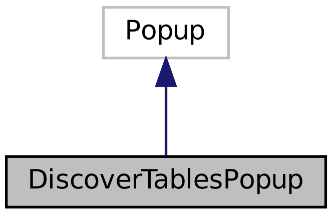
The documentation for this class was generated from the following file:

- ViewModels/Popups/CustomStatisticsViewModel.cs

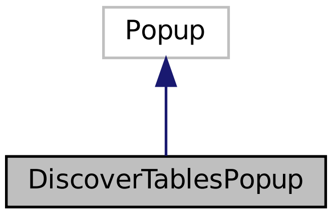
3.18 DiscoverTablesPopup Class Reference

Popup for displaying the interfaces for the discover tables function

Inheritance diagram for DiscoverTablesPopup:



Collaboration diagram for DiscoverTablesPopup:



Public Member Functions

- DiscoverTablesPopup (DiscoverTablesViewModel discoveredTableViewModel)

  The view model for the discovered tables popup.

3.18.1 Detailed Description

Popup for displaying the interfaces for the discover tables function

Author

Tormod Smidesang

Elvin Andreas Pedersen

3.18.2 Constructor & Destructor Documentation

3.18.2.1 DiscoverTablesPopup()

```
DiscoverTablesPopup.DiscoverTablesPopup (
            DiscoverTablesViewModel discoveredTableViewModel )  [inline]
```

The view model for the discovered tables popup.

Parameters

discoveredTableViewModel	Dependency injection of the view model for the discovered tables popup.

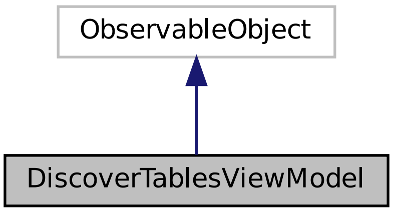
The documentation for this class was generated from the following file:

- Views/Popups/DiscoverTablesPopup.xaml.cs

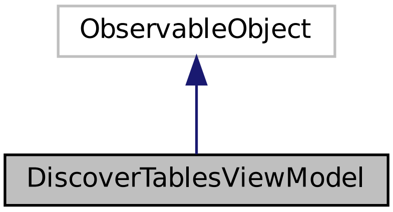
3.19 DiscoverTablesViewModel Class Reference

The viewmodel for the discover tables popup

Inheritance diagram for DiscoverTablesViewModel:



Collaboration diagram for DiscoverTablesViewModel:



Public Member Functions

- DiscoverTablesViewModel (TableScanService tableScanService, TableContextService tableContextService)

  Constructor. Sets members and populates the list of network interfaces
- void DiscoverTables ()

  Called when the user clicks the discover tables button. Initiates a scan on the selected network interface.
- void AbortScan ()

  Called when the user clicks on the abort scan button. Cancels a scan in progress.

Properties

- ObservableCollection< NetworkInterface > Interfaces = [ ] `[get]`

  List of network interfaces, excluding interfaces connected to networks that are not class C

3.19.1 Detailed Description

The viewmodel for the discover tables popup

Author

Tormod Smidesang

Elvin Andreas Pedersen

3.19.2 Constructor & Destructor Documentation

3.19.2.1 DiscoverTablesViewModel()

```
DiscoverTablesViewModel.DiscoverTablesViewModel (
            TableScanService tableScanService,
            TableContextService tableContextService )  [inline]
```

Constructor. Sets members and populates the list of network interfaces

Parameters

tableScanService	The table scan service
tableContextService	The table context service

3.19.3 Member Function Documentation

3.19.3.1 AbortScan()

```
void DiscoverTablesViewModel.AbortScan ( )  [inline]
```

Called when the user clicks on the abort scan button. Cancels a scan in progress.

3.19.3.2 DiscoverTables()

```
void DiscoverTablesViewModel.DiscoverTables ( )  [inline]
```

Called when the user clicks the discover tables button. Initiates a scan on the selected network interface.

3.19.4 Property Documentation

3.19.4.1 Interfaces

```
ObservableCollection<NetworkInterface> DiscoverTablesViewModel.Interfaces = [ ]  [get]
```

List of network interfaces, excluding interfaces connected to networks that are not class C

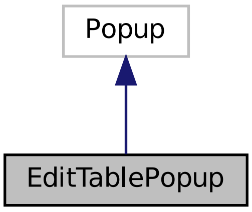
The documentation for this class was generated from the following file:

- ViewModels/Popups/DiscoverTablesViewModel.cs

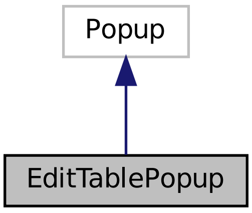
3.20 EditTablePopup Class Reference

Popup for editing a table.

Inheritance diagram for EditTablePopup:



Collaboration diagram for EditTablePopup:



Public Member Functions

- EditTablePopup (EditTableViewModel editTableViewModel)

  Initializes a new instance of the EditTablePopup class.

Properties

- Command CloseCommand `[get]`

  Command to close the popup.

3.20.1 Detailed Description

Popup for editing a table.

Author

Ole William Skistad Huslende

Tormod Smidesang

Elvin Andreas Pedersen

3.20.2 Constructor & Destructor Documentation

3.20.2.1 EditTablePopup()

```
EditTablePopup.EditTablePopup (
            EditTableViewModel editTableViewModel )  [inline]
```

Initializes a new instance of the EditTablePopup class.

Parameters

editTableViewModel	Dependency injection of the view model for the edit table popup.

3.20.3 Property Documentation

3.20.3.1 CloseCommand

```
Command EditTablePopup.CloseCommand  [get]
```

Command to close the popup.

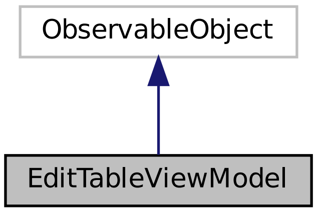
The documentation for this class was generated from the following file:

- Views/Popups/EditTablePopup.xaml.cs

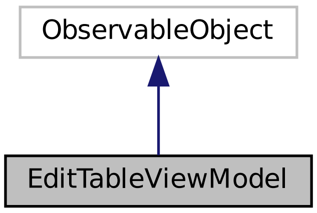
3.21 EditTableViewModel Class Reference

The view model for the edit table popup.

Inheritance diagram for EditTableViewModel:



Collaboration diagram for EditTableViewModel:



Public Member Functions

- EditTableViewModel (TableContextService tableContextService, TableEntryFieldValidator tableEntryField↩ Validator)

  Initializes a new instance of the EditTableViewModel class.
- void SetPopup (EditTablePopup Popup)

  Sets the popup for editing the table.

Properties

- string NameError `[get]`

  The error message for the table name field.
- string IpError `[get]`

  The error message for the IP address field.
- string ClientIdError `[get]`

  The error message for the client ID field.
- string ClientSecretError `[get]`

  The error message for the client secret field.
- bool IsFormValid `[get]`

  Checks if the form is valid.

3.21.1 Detailed Description

The view model for the edit table popup.

Handle the editing of the table information.

Author

Ole William Skistad Huslende

Tormod Smidesang

Elvin Andreas Pedersen

3.21.2 Constructor & Destructor Documentation

3.21.2.1 EditTableViewModel()

```
EditTableViewModel.EditTableViewModel (
            TableContextService tableContextService,
            TableEntryFieldValidator tableEntryFieldValidator )  [inline]
```

Initializes a new instance of the EditTableViewModel class.

Parameters

tableContextService	The service for managing the table contexts.
tableEntryFieldValidator	The validator for the table entry fields.

3.21.3 Member Function Documentation

3.21.3.1 SetPopup()

```
void EditTableViewModel.SetPopup (
            EditTablePopup Popup )  [inline]
```

Sets the popup for editing the table.

Parameters

Popup	

3.21.4 Property Documentation

3.21.4.1 ClientIdError

```
string EditTableViewModel.ClientIdError  [get]
```

The error message for the client ID field.

3.21.4.2 ClientSecretError

```
string EditTableViewModel.ClientSecretError  [get]
```

The error message for the client secret field.

3.21.4.3 IpError

```
string EditTableViewModel.IpError  [get]
```

The error message for the IP address field.

3.21.4.4 IsFormValid

```
bool EditTableViewModel.IsFormValid  [get]
```

Checks if the form is valid.

3.21.4.5 NameError

```
string EditTableViewModel.NameError  [get]
```

The error message for the table name field.

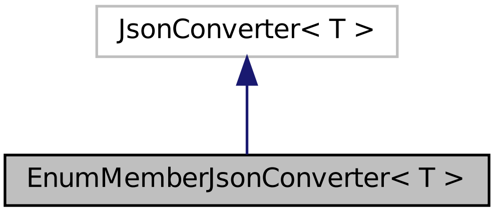
The documentation for this class was generated from the following file:

- ViewModels/Popups/EditTableViewModel.cs

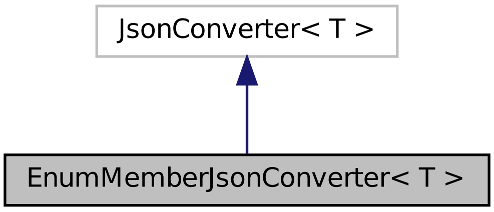
3.22 EnumMemberJsonConverter< T > Class Template Reference

This class is a custom JSON converter for enums that use the EnumMember attribute.

Inheritance diagram for EnumMemberJsonConverter< T >:



Collaboration diagram for EnumMemberJsonConverter< T >:



Public Member Functions

- override T Read (ref Utf8JsonReader reader, Type typeToConvert, JsonSerializerOptions options)

  Reads the JSON value and converts it to the enum type.
- override void Write (Utf8JsonWriter writer, T value, JsonSerializerOptions options)

  Writes the enum value as a JSON string using the EnumMember attribute value if it exists.

3.22.1 Detailed Description

This class is a custom JSON converter for enums that use the EnumMember attribute.

Template Parameters

T	The type of the enum. It must be a struct and an enum.

Author

Elvin Andreas Pedersen

Type Constraints

T* : *struct

T* : *Enum

3.22.2 Member Function Documentation

3.22.2.1 Read()

```
override T EnumMemberJsonConverter< T >.Read (
             ref Utf8JsonReader reader,
             Type typeToConvert,
             JsonSerializerOptions options )  [inline]
```

Reads the JSON value and converts it to the enum type.

Parameters

reader	The JSON reader that reads the JSON value.
typeToConvert	The type to convert to. This is the enum type.
options	The JSON serializer options.

Returns

The enum value that corresponds to the JSON value.

3.22.2.2 Write()

```
override void EnumMemberJsonConverter< T >.Write (
            Utf8JsonWriter writer,
            T value,
            JsonSerializerOptions options )  [inline]
```

Writes the enum value as a JSON string using the EnumMember attribute value if it exists.

Parameters

writer	The JSON writer that writes the JSON value.
value	The enum value to write.
options	The JSON serializer options.

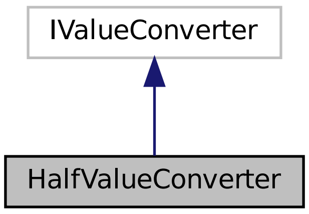
The documentation for this class was generated from the following file:

- Utilities/EnumMemberJsonConverter.cs

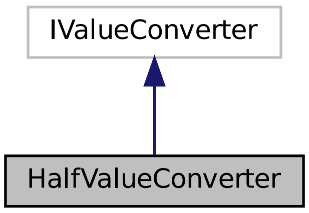
3.23 HalfValueConverter Class Reference

Used in xaml to get half of a value.

Inheritance diagram for HalfValueConverter:



Collaboration diagram for HalfValueConverter:



Public Member Functions

- object? Convert (object? value, Type targetType, object? parameter, CultureInfo culture)

  Halves a value.
- object? ConvertBack (object? value, Type targetType, object? parameter, CultureInfo culture)

  Not implemented.

3.23.1 Detailed Description

Used in xaml to get half of a value.

Author

Tormod Smidesang

3.23.2 Member Function Documentation

3.23.2.1 Convert()

```
object? HalfValueConverter.Convert (
            object? value,
            Type targetType,
            object? parameter,
            CultureInfo culture ) [inline]
```

Halves a value.

Parameters

value	The value to halve
targetType	unused
parameter	unused
culture	unused

Returns

Exceptions

NotImplementedException	

3.23.2.2 ConvertBack()

```
object?  HalfValueConverter.ConvertBack (
             object?  value,
             Type targetType,
             object?  parameter,
             CultureInfo culture )  [inline]
```

Not implemented.

The documentation for this class was generated from the following file:

- Utilities/Converters/HalfValueConverter.cs

3.24 HistoryEntryAddedEvent Class Reference

Represents an event from the Kongsberg HUB where a job is added to the history queue.

Properties

- string ID `[get, set]`
  The Id of the job that was added to the history queue.
- EventAction Action `[get, set]`
  The action that triggered the event EventAction.

3.24.1 Detailed Description

Represents an event from the Kongsberg HUB where a job is added to the history queue.

Author

Ole William Skistad Huslende

3.24.2 Property Documentation

3.24.2.1 Action

```
EventAction HistoryEntryAddedEvent.Action  [get], [set]
```

The action that triggered the event EventAction.

3.24.2.2 ID

```
string HistoryEntryAddedEvent.ID  [get], [set]
```

The Id of the job that was added to the history queue.

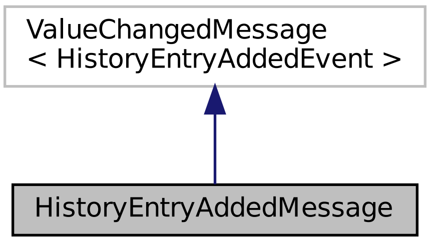
The documentation for this class was generated from the following file:

- Models/ApiSSE/EventTypes.cs

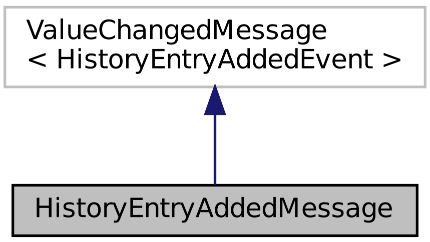
3.25 HistoryEntryAddedMessage Class Reference

Message to send history entry added event received from the Kongsberg HUB and propagate it forward to the viewmodels.

Inheritance diagram for HistoryEntryAddedMessage:



Collaboration diagram for HistoryEntryAddedMessage:



Public Member Functions

- **HistoryEntryAddedMessage** (HistoryEntryAddedEvent value)

3.25.1 Detailed Description

Message to send history entry added event received from the Kongsberg HUB and propagate it forward to the viewmodels.

This class is used to send messages between the model and the viewmodel with the help of IMessenger.

Author

Ole William Skistad Huslende

The documentation for this class was generated from the following file:

- Models/ModelMessages.cs

3.26 HistoryEntryRemovedEvent Class Reference

Represents an event from the Kongsberg HUB where a job is removed from the history queue.

Properties

- string ID `[get, set]`

  The Id of the job that was removed from the history queue.
- EventAction Action `[get, set]`

  The action that triggered the event EventAction.

3.26.1 Detailed Description

Represents an event from the Kongsberg HUB where a job is removed from the history queue.

Author

Ole William Skistad Huslende

3.26.2 Property Documentation

3.26.2.1 Action

```
EventAction HistoryEntryRemovedEvent.Action  [get], [set]
```

The action that triggered the event EventAction.

3.26.2.2 ID

```
string HistoryEntryRemovedEvent.ID  [get], [set]
```

The Id of the job that was removed from the history queue.

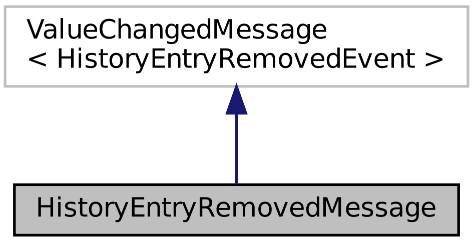
The documentation for this class was generated from the following file:

- Models/ApiSSE/EventTypes.cs

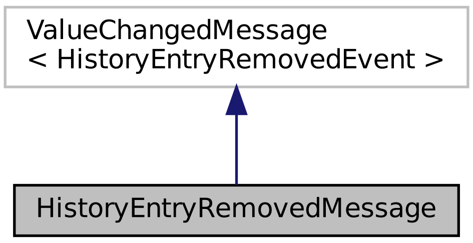
3.27 HistoryEntryRemovedMessage Class Reference

Message to send history entry removed event received from the Kongsberg HUB and propagate it forward to the viewmodels.

Inheritance diagram for HistoryEntryRemovedMessage:



Collaboration diagram for HistoryEntryRemovedMessage:



Public Member Functions

- **HistoryEntryRemovedMessage** (HistoryEntryRemovedEvent value)

3.27.1 Detailed Description

Message to send history entry removed event received from the Kongsberg HUB and propagate it forward to the viewmodels.

This class is used to send messages between the model and the viewmodel with the help of IMessenger.

Author

Ole William Skistad Huslende

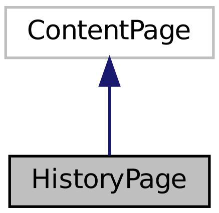
The documentation for this class was generated from the following file:

- Models/ModelMessages.cs

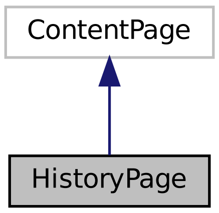
3.28 HistoryPage Class Reference

HistoryPage is the page that shows the history of jobs for a table.

Inheritance diagram for HistoryPage:



Collaboration diagram for HistoryPage:



Public Member Functions

- HistoryPage (TableContextService tableContextService, InfoPanelContentView infoPanelContentView)

  The constructor for the HistoryPage.

Protected Member Functions

- override void OnAppearing ()

  Called when the page is appearing. Used to set the binding context for the page and the info panel. Also resets the selected job in the history view model to null.

3.28.1 Detailed Description

HistoryPage is the page that shows the history of jobs for a table.

Author

Ole William Skistad Huslende

Tormod Smidesang

Elvin Andreas Pedersen

3.28.2 Constructor & Destructor Documentation

3.28.2.1 HistoryPage()

```
HistoryPage.HistoryPage (
            TableContextService tableContextService,
            InfoPanelContentView infoPanelContentView )  [inline]
```

The constructor for the HistoryPage.

Parameters

tableContextService	Dependency injection for the table context service.
infoPanelContentView	Dependency injection for the info panel content view.

3.28.3 Member Function Documentation

3.28.3.1 OnAppearing()

```
override void HistoryPage.OnAppearing ( )  [inline], [protected]
```

Called when the page is appearing. Used to set the binding context for the page and the info panel. Also resets the selected job in the history view model to null.

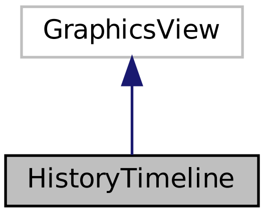
The documentation for this class was generated from the following file:

- Views/TableContextViews/HistoryPage.xaml.cs

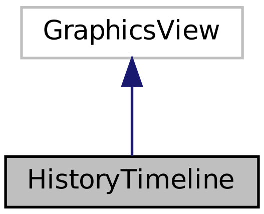
3.29 HistoryTimeline Class Reference

Custom control for displaying a timeline of job runs shown on statistics page and meta statistics page.

Inheritance diagram for HistoryTimeline:



Collaboration diagram for HistoryTimeline:



Public Member Functions

- HistoryTimeline ()

  Initializes a new instance of the HistoryTimeline class.

Static Public Attributes

- static readonly BindableProperty StartProperty

    Bindable property for the start date of the timeline.
- static readonly BindableProperty EndProperty

    Bindable property for the end date of the timeline.
- static readonly BindableProperty ItemsProperty

    Bindable property for the list of job statistics.
- static readonly BindableProperty SelectedJobItemProperty

    Bindable property for the selected job item.
- static readonly BindableProperty TimelineColorProperty

    Bindable property for the color of the timeline.
- static readonly BindableProperty VerticalLineColorProperty

    Bindable property for the color of the vertical line.
- static readonly BindableProperty ItemColorProperty

    Bindable property for the color of the items.
- static readonly BindableProperty SelectedColorProperty

    Bindable property for the color of the selected item.

Properties

- DateTime Start `[get, set]`

    Gets or sets the start date of the timeline.
- DateTime End `[get, set]`

    Gets or sets the end date of the timeline.
- ObservableCollection< JobStatisticsViewModel > Items `[get, set]`

    Gets or sets the list of job statistics.
- JobStatisticsViewModel? SelectedJobItem `[get, set]`

    Gets or sets the selected job item.
- Color TimelineColor `[get, set]`

    Gets or sets the color of the timeline.
- Color VerticalLineColor `[get, set]`

    Gets or sets the color of the vertical line.
- Color ItemColor `[get, set]`

    Gets or sets the color of the items.
- Color SelectedColor `[get, set]`

    Gets or sets the color of the selected item.

3.29.1 Detailed Description

Custom control for displaying a timeline of job runs shown on statistics page and meta statistics page.

Author

Ole William Skistad Huslende

3.29.2 Constructor & Destructor Documentation

3.29.2.1 HistoryTimeline()

```
HistoryTimeline.HistoryTimeline ( )  [inline]
```

Initializes a new instance of the HistoryTimeline class.

3.29.3 Member Data Documentation

3.29.3.1 EndProperty

```
readonly BindableProperty HistoryTimeline.EndProperty  [static]
```

Initial value:

```
=
    BindableProperty.Create(
        nameof(End),
        typeof(DateTime),
        typeof(HistoryTimeline),
        default)
```

Bindable property for the end date of the timeline.

3.29.3.2 ItemColorProperty

```
readonly BindableProperty HistoryTimeline.ItemColorProperty  [static]
```

Initial value:

```
=
       BindableProperty.Create(
           nameof(ItemColor),
           typeof(Color),
           typeof(HistoryTimeline),
           Colors.Orange,
           propertyChanged: OnPropertyChanged)
```

Bindable property for the color of the items.

3.29.3.3 ItemsProperty

```
readonly BindableProperty HistoryTimeline.ItemsProperty  [static]
```

Initial value:

```
=
        BindableProperty.Create(
            nameof(Items),
            typeof(ObservableCollection<JobStatisticsViewModel>),
            typeof(HistoryTimeline),
            default,
            propertyChanged: OnItemsPropertyChanged)
```

Bindable property for the list of job statistics.

3.29.3.4 SelectedColorProperty

```
readonly BindableProperty HistoryTimeline.SelectedColorProperty [static]
```

Initial value:

```
=
        BindableProperty.Create(
            nameof(SelectedColor),
            typeof(Color),
            typeof(HistoryTimeline),
            Colors.Gray,
            propertyChanged: OnPropertyChanged)
```

Bindable property for the color of the selected item.

3.29.3.5 SelectedJobItemProperty

```
readonly BindableProperty HistoryTimeline.SelectedJobItemProperty [static]
```

Initial value:

```
=
      BindableProperty.Create(
          nameof(SelectedJobItem),
          typeof(JobStatisticsViewModel),
          typeof(HistoryTimeline),
          default,
          propertyChanged: OnPropertyChanged)
```

Bindable property for the selected job item.

3.29.3.6 StartProperty

```
readonly BindableProperty HistoryTimeline.StartProperty [static]
```

Initial value:

```
=
        BindableProperty.Create(
            nameof(Start),
            typeof(DateTime),
            typeof(HistoryTimeline),
            default)
```

Bindable property for the start date of the timeline.

3.29.3.7 TimelineColorProperty

```
readonly BindableProperty HistoryTimeline.TimelineColorProperty [static]
```

Initial value:

```
=
        BindableProperty.Create(
            nameof(TimelineColor),
            typeof(Color),
            typeof(HistoryTimeline),
            Colors.Orange,
            propertyChanged: OnPropertyChanged)
```

Bindable property for the color of the timeline.

3.29.3.8 VerticalLineColorProperty

```
readonly BindableProperty HistoryTimeline.VerticalLineColorProperty  [static]
```

Initial value:

```
=
       BindableProperty.Create(
           nameof(VerticalLineColor),
           typeof(Color),
           typeof(HistoryTimeline),
           Colors.Orange,
           propertyChanged: OnPropertyChanged)
```

Bindable property for the color of the vertical line.

3.29.4 Property Documentation

3.29.4.1 End

```
DateTime HistoryTimeline.End  [get], [set]
```

Gets or sets the end date of the timeline.

3.29.4.2 ItemColor

```
Color HistoryTimeline.ItemColor  [get], [set]
```

Gets or sets the color of the items.

3.29.4.3 Items

```
ObservableCollection<JobStatisticsViewModel> HistoryTimeline.Items  [get], [set]
```

Gets or sets the list of job statistics.

3.29.4.4 SelectedColor

```
Color HistoryTimeline.SelectedColor  [get], [set]
```

Gets or sets the color of the selected item.

3.29.4.5 SelectedJobItem

```
JobStatisticsViewModel? HistoryTimeline.SelectedJobItem [get], [set]
```

Gets or sets the selected job item.

3.29.4.6 Start

```
DateTime HistoryTimeline.Start [get], [set]
```

Gets or sets the start date of the timeline.

3.29.4.7 TimelineColor

```
Color HistoryTimeline.TimelineColor [get], [set]
```

Gets or sets the color of the timeline.

3.29.4.8 VerticalLineColor

```
Color HistoryTimeline.VerticalLineColor [get], [set]
```

Gets or sets the color of the vertical line.

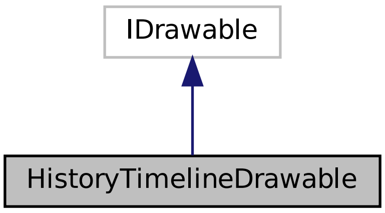
The documentation for this class was generated from the following file:

- Views/Components/HistoryTimeLine.cs

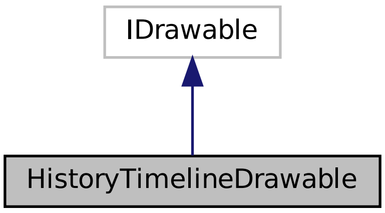
3.30 HistoryTimelineDrawable Class Reference

Custom drawable for the HistoryTimeline.

Inheritance diagram for HistoryTimelineDrawable:



Collaboration diagram for HistoryTimelineDrawable:



Public Member Functions

- HistoryTimelineDrawable (HistoryTimeline timeline)

  Initializes a new instance of the HistoryTimelineDrawable class.
- void Draw (ICanvas canvas, RectF dirtyRect)

  Draws the timeline on the canvas.

3.30.1 Detailed Description

Custom drawable for the HistoryTimeline.

3.30.2 Constructor & Destructor Documentation

3.30.2.1 HistoryTimelineDrawable()

```
HistoryTimelineDrawable.HistoryTimelineDrawable (
            HistoryTimeline timeline )  [inline]
```

Initializes a new instance of the HistoryTimelineDrawable class.

Parameters

timeline	The history timeline instance.

3.30.3 Member Function Documentation

3.30.3.1 Draw()

```
void HistoryTimelineDrawable.Draw (
            ICanvas canvas,
            RectF dirtyRect )  [inline]
```

Draws the timeline on the canvas.

Parameters

canvas	The canvas to draw on.
dirtyRect	The rectangle that needs to be redrawn.

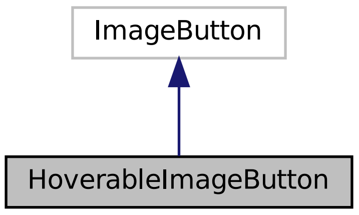
The documentation for this class was generated from the following file:

- Views/Components/HistoryTimeLine.cs

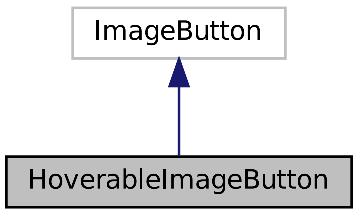
3.31 HoverableImageButton Class Reference

A custom ImageButton that changes its appearance when hovered over.

Inheritance diagram for HoverableImageButton:



Collaboration diagram for HoverableImageButton:



Public Member Functions

- HoverableImageButton ()

  Initializes a new instance of the HoverableImageButton class.

Static Public Attributes

- static readonly BindableProperty HoverOpacityProperty

  Bindable property for the hover opacity of the button.
- static readonly BindableProperty HoverBorderColorProperty

  Bindable property for the hover border color of the button.
- static readonly BindableProperty HoverBorderWidthProperty

  Bindable property for the hover border width of the button.
- static readonly BindableProperty HoverCornerRadiusProperty

  Bindable property for the hover corner radius of the button.

Properties

- double HoverOpacity `[get, set]`

  Gets or sets the hover opacity of the button.
- Color HoverBorderColor `[get, set]`

  Gets or sets the hover border color of the button.
- int HoverBorderWidth `[get, set]`

  Gets or sets the hover border width of the button.
- int HoverCornerRadius `[get, set]`

  Gets or sets the hover corner radius of the button.

3.31.1 Detailed Description

A custom ImageButton that changes its appearance when hovered over.

Author

Elvin Andreas Pedersen

3.31.2 Constructor & Destructor Documentation

3.31.2.1 HoverableImageButton()

```
HoverableImageButton.HoverableImageButton ( ) [inline]
```

Initializes a new instance of the HoverableImageButton class.

3.31.3 Member Data Documentation

3.31.3.1 HoverBorderColorProperty

```
readonly BindableProperty HoverableImageButton.HoverBorderColorProperty [static]
```

Initial value:

```
=
        BindableProperty.Create(nameof(HoverBorderColor), typeof(Color), typeof(HoverableImageButton),
       Colors.LightGray)
```

Bindable property for the hover border color of the button.

3.31.3.2 HoverBorderWidthProperty

```
readonly BindableProperty HoverableImageButton.HoverBorderWidthProperty [static]
```

Initial value:

```
=
        BindableProperty.Create(nameof(HoverBorderWidth), typeof(int), typeof(HoverableImageButton), 1)
```

Bindable property for the hover border width of the button.

3.31.3.3 HoverCornerRadiusProperty

```
readonly BindableProperty HoverableImageButton.HoverCornerRadiusProperty [static]
```

Initial value:

```
=
        BindableProperty.Create(nameof(HoverCornerRadius), typeof(int), typeof(HoverableImageButton), 5)
```

Bindable property for the hover corner radius of the button.

3.31.3.4 HoverOpacityProperty

```
readonly BindableProperty HoverableImageButton.HoverOpacityProperty [static]
```

Initial value:

```
=
        BindableProperty.Create(nameof(HoverOpacity), typeof(double), typeof(HoverableImageButton), 0.9)
```

Bindable property for the hover opacity of the button.

3.31.4 Property Documentation

3.31.4.1 HoverBorderColor

```
Color HoverableImageButton.HoverBorderColor [get], [set]
```

Gets or sets the hover border color of the button.

3.31.4.2 HoverBorderWidth

```
int HoverableImageButton.HoverBorderWidth [get], [set]
```

Gets or sets the hover border width of the button.

3.31.4.3 HoverCornerRadius

```
int HoverableImageButton.HoverCornerRadius  [get], [set]
```

Gets or sets the hover corner radius of the button.

3.31.4.4 HoverOpacity

```
double HoverableImageButton.HoverOpacity  [get], [set]
```

Gets or sets the hover opacity of the button.

The documentation for this class was generated from the following file:

- Utilities/HoverBehaviour.cs

3.32 IdentificationField Class Reference

Represents a field containing identification information of barcode.

Properties

- string? Value `[get, set]`

  Gets or sets the value of the barcode.
- string? EncodingDetails `[get, set]`

  Gets or sets type of barcode.
- List< double >? BoundingBox `[get, set]`

  Gets or sets the position of the barcode.
- List< double >? Matrix `[get, set]`

  Gets or sets the orientation of the barcode.

3.32.1 Detailed Description

Represents a field containing identification information of barcode.

Author

Ole William Skistad Huslende

3.32.2 Property Documentation

3.32.2.1 BoundingBox

```
List<double>?  IdentificationField.BoundingBox  [get], [set]
```

Gets or sets the position of the barcode.

3.32.2.2 EncodingDetails

```
string?  IdentificationField.EncodingDetails  [get], [set]
```

Gets or sets type of barcode.

3.32.2.3 Matrix

```
List<double>?  IdentificationField.Matrix  [get], [set]
```

Gets or sets the orientation of the barcode.

3.32.2.4 Value

```
string?  IdentificationField.Value  [get], [set]
```

Gets or sets the value of the barcode.

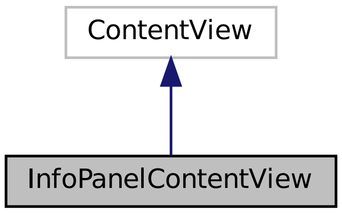
The documentation for this class was generated from the following file:

- Models/ApiQueue/IdentificationField.cs

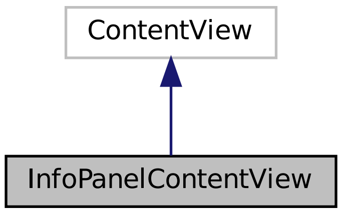
3.33 InfoPanelContentView Class Reference

This class is the content view for the info panel.

Inheritance diagram for InfoPanelContentView:



Collaboration diagram for InfoPanelContentView:



3.33.1 Detailed Description

This class is the content view for the info panel.

Author

Ole William Skistad Huslende

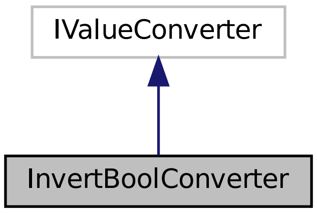
The documentation for this class was generated from the following file:

- Views/TableContextViews/InfoPanelContentView.xaml.cs

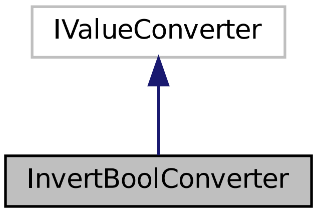
3.34 InvertBoolConverter Class Reference

Used in xaml to invert a bool.

Inheritance diagram for InvertBoolConverter:



Collaboration diagram for InvertBoolConverter:



Public Member Functions

- object? Convert (object? value, Type targetType, object? parameter, CultureInfo culture)

    Inverts a boolean.
- object? ConvertBack (object? value, Type targetType, object? parameter, CultureInfo culture)

    Simply calls Convert(object?, Type, object?, CultureInfo) because converting an inverted bool back is the same operation

3.34.1 Detailed Description

Used in xaml to invert a bool.

Author

Tormod Smidesang

3.34.2 Member Function Documentation

3.34.2.1 Convert()

```
object?  InvertBoolConverter.Convert (
             object?  value,
             Type targetType,
             object?  parameter,
             CultureInfo culture )  [inline]
```

Inverts a boolean.

Parameters

value	The bool to invert
targetType	unused
parameter	unused
culture	unused

Returns

The inverted bool

Exceptions

ArgumentException	Thrown if the provided value is not a bool

3.34.2.2 ConvertBack()

```
object?  InvertBoolConverter.ConvertBack (
             object?  value,
             Type targetType,
             object?  parameter,
             CultureInfo culture )  [inline]
```

Simply calls Convert(object?, Type, object?, CultureInfo) because converting an inverted bool back is the same operation

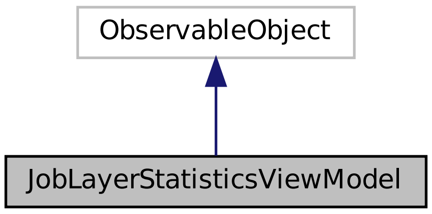
The documentation for this class was generated from the following file:

- Utilities/Converters/InvertBoolConverter.cs

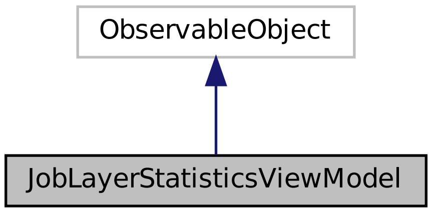
3.35 JobLayerStatisticsViewModel Class Reference

ViewModel for displaying job layer statistics.

Inheritance diagram for JobLayerStatisticsViewModel:



Collaboration diagram for JobLayerStatisticsViewModel:



Public Member Functions

- JobLayerStatisticsViewModel (IServiceProvider serviceProvider, IMessenger messenger, QueueServerLayerStatistics jobLayerStatistics)
  Initializes a new instance of the JobLayerStatisticsViewModel class.
- void UpdateJobLayerStatistics ()
  Updates the job layer statistics properties with the values from the model.

3.35.1 Detailed Description

ViewModel for displaying job layer statistics.

This class is responsible for managing the job layer statistics data and providing properties to bind to the UI. Only the important properties are included from the model.

Author

Ole William Skistad Huslende

3.35.2 Constructor & Destructor Documentation

3.35.2.1 JobLayerStatisticsViewModel()

```
JobLayerStatisticsViewModel.JobLayerStatisticsViewModel (
             IServiceProvider serviceProvider,
             IMessenger messenger,
             QueueServerLayerStatistics jobLayerStatistics )  [inline]
```

Initializes a new instance of the JobLayerStatisticsViewModel class.

Parameters

serviceProvider	The service provider used to resolve dependencies.
messenger	The messenger used for communication inside the table context.
jobLayerStatistics	The job layer statistics model that this ViewModel is based on.

3.35.3 Member Function Documentation

3.35.3.1 UpdateJobLayerStatistics()

```
void JobLayerStatisticsViewModel.UpdateJobLayerStatistics ( )  [inline]
```

Updates the job layer statistics properties with the values from the model.

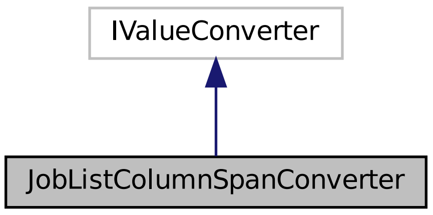
The documentation for this class was generated from the following file:

- ViewModels/TableContextViewModels/SubViewModels/JobLayerStatisticsViewModel.cs

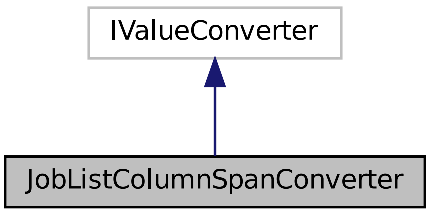
3.36 JobListColumnSpanConverter Class Reference

Used to expand the job list to cover the area used by the job details panel when the latter is not shown.

Inheritance diagram for JobListColumnSpanConverter:



Collaboration diagram for JobListColumnSpanConverter:



Public Member Functions

- object? Convert (object? value, Type targetType, object? parameter, CultureInfo culture)

  Returns how many columns the job list should cover based on whether or not the job details panel is visible.
- object? ConvertBack (object? value, Type targetType, object? parameter, CultureInfo culture)

  Not implemented.

3.36.1 Detailed Description

Used to expand the job list to cover the area used by the job details panel when the latter is not shown.

Author

Tormod Smidesang

3.36.2 Member Function Documentation

3.36.2.1 Convert()

```
object? JobListColumnSpanConverter.Convert (
            object? value,
            Type targetType,
            object? parameter,
            CultureInfo culture ) [inline]
```

Returns how many columns the job list should cover based on whether or not the job details panel is visible.

Parameters

value	Whether or not the job details panel is visible
targetType	unused
parameter	unused
culture	unused

Returns

How many columns the job list should cover, or null if parameter value is not a bool

3.36.2.2 ConvertBack()

```
object?  JobListColumnSpanConverter.ConvertBack (
             object?  value,
             Type targetType,
             object?  parameter,
             CultureInfo culture )  [inline]
```

Not implemented.

The documentation for this class was generated from the following file:

- Utilities/Converters/JobListColumnSpanConverter.cs

3.37 JobQueue Class Reference

The model for the list of jobs from the table.

Properties

- string? Id `[get, set]`
  Gets or sets the Id for the job queue.
- List< JobTask >? JobTasks `[get, set]`
  Gets or sets the list of jobs JobTask

3.37.1 Detailed Description

The model for the list of jobs from the table.

Author

Ole William Skistad Huslende

3.37.2 Property Documentation

3.37.2.1 Id

```
string?  JobQueue.Id  [get], [set]
```

Gets or sets the Id for the job queue.

3.37.2.2 JobTasks

```
List<JobTask>?  JobQueue.JobTasks  [get], [set]
```

Gets or sets the list of jobs JobTask

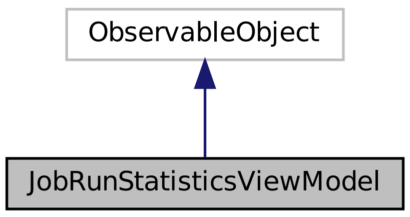
The documentation for this class was generated from the following file:

- Models/ApiQueue/JobQueue.cs

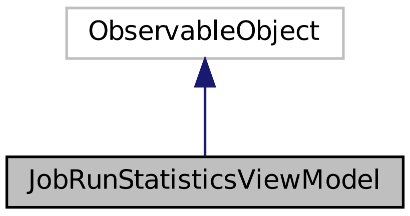
3.38 JobRunStatisticsViewModel Class Reference

A view model for job run statistics.

Inheritance diagram for JobRunStatisticsViewModel:



Collaboration diagram for JobRunStatisticsViewModel:



Public Member Functions

- JobRunStatisticsViewModel (IServiceProvider serviceProvider, IMessenger messenger, QueueServerProductionRunStatistics jobRunStatistics)

  Initializes a new instance of the JobRunStatisticsViewModel class.
- void UpdateJobRunStatistics ()

  Updates the job run statistics with the data from the jobRunStatistics object.

Properties

- ObservableCollection< JobLayerStatisticsViewModel > JobLayerStatistics = new() `[get, set]`

  List of job layer statistics associated with the job run.

3.38.1 Detailed Description

A view model for job run statistics.

This class is used to represent the statistics of a job run. Only the important properties are included from the model.

Author

Ole William Skistad Huslende

3.38.2 Constructor & Destructor Documentation

3.38.2.1 JobRunStatisticsViewModel()

```
JobRunStatisticsViewModel.JobRunStatisticsViewModel (
            IServiceProvider serviceProvider,
            IMessenger messenger,
            QueueServerProductionRunStatistics jobRunStatistics )  [inline]
```

Initializes a new instance of the JobRunStatisticsViewModel class.

Parameters

serviceProvider	The service provider used to create instances of viewmodels.
messenger	The messenger used for communication between view models inside the table context TableContext.
jobRunStatistics	The job run statistics data model QueueServerProductionRunStatistics.

3.38.3 Member Function Documentation

3.38.3.1 UpdateJobRunStatistics()

```
void JobRunStatisticsViewModel.UpdateJobRunStatistics ( )  [inline]
```

Updates the job run statistics with the data from the jobRunStatistics object.

3.38.4 Property Documentation

3.38.4.1 JobLayerStatistics

```
ObservableCollection<JobLayerStatisticsViewModel> JobRunStatisticsViewModel.JobLayerStatistics
= new()  [get], [set]
```

List of job layer statistics associated with the job run.

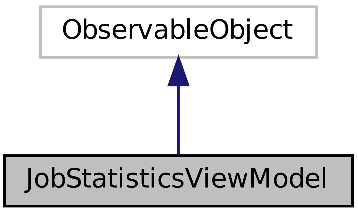
The documentation for this class was generated from the following file:

- ViewModels/TableContextViewModels/SubViewModels/JobRunStatisticsViewModel.cs

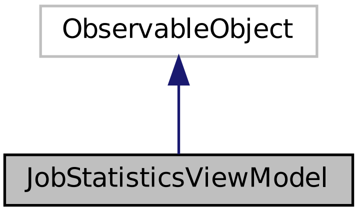
3.39 JobStatisticsViewModel Class Reference

ViewModel for displaying job statistics.

Inheritance diagram for JobStatisticsViewModel:



Collaboration diagram for JobStatisticsViewModel:



Public Member Functions

- JobStatisticsViewModel (IServiceProvider serviceProvider, IMessenger messenger, QueueServerJobStatistics jobStatistics, string tableName)

  Initializes a new instance of the JobStatisticsViewModel class.
- void UpdateJobStatistics ()

  Updates the job statistics with the data from the job statistics instance.

Properties

- ObservableCollection< JobRunStatisticsViewModel > JobRuns = [ ] `[get, set]`

  A collection of the job runs associated with the job statistics JobRunStatisticsViewModel.

3.39.1 Detailed Description

ViewModel for displaying job statistics.

This class is responsible for managing and displaying job statistics data.

Author

Ole William Skistad Huslende

3.39.2 Constructor & Destructor Documentation

3.39.2.1 JobStatisticsViewModel()

```
JobStatisticsViewModel.JobStatisticsViewModel (
            IServiceProvider serviceProvider,
            IMessenger messenger,
            QueueServerJobStatistics jobStatistics,
            string tableName )  [inline]
```

Initializes a new instance of the JobStatisticsViewModel class.

Parameters

serviceProvider	The service provider instance used for dependency injection IServiceProvider.
messenger	The messenger instance used for communication between view models inside the table context TableContext.
jobStatistics	The job statistics instance that holds the data QueueServerJobStatistics.
tableName	The table the job statistics belongs to.

3.39.3 Member Function Documentation

3.39.3.1 UpdateJobStatistics()

```
void JobStatisticsViewModel.UpdateJobStatistics ( )  [inline]
```

Updates the job statistics with the data from the job statistics instance.

3.39.4 Property Documentation

3.39.4.1 JobRuns

```
ObservableCollection<JobRunStatisticsViewModel> JobStatisticsViewModel.JobRuns = [ ]  [get],
[set]
```

A collection of the job runs associated with the job statistics JobRunStatisticsViewModel.

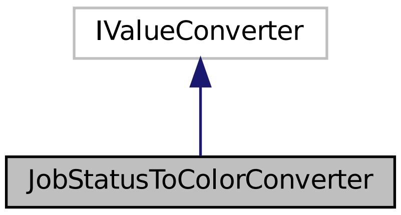
The documentation for this class was generated from the following file:

- ViewModels/TableContextViewModels/SubViewModels/JobStatisticsViewModel.cs

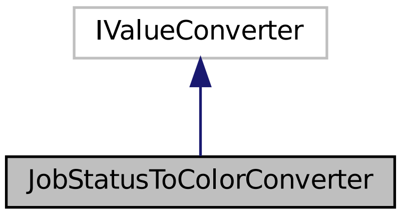
3.40 JobStatusToColorConverter Class Reference

Gets a color associated with a given job status.

Inheritance diagram for JobStatusToColorConverter:



Collaboration diagram for JobStatusToColorConverter:



Public Member Functions

- object? Convert (object? value, Type targetType, object? parameter, CultureInfo culture)

  Converts a job status to a color.
- object? ConvertBack (object? value, Type targetType, object? parameter, CultureInfo culture)

  Not implemented.

3.40.1 Detailed Description

Gets a color associated with a given job status.

Author

Tormod Smidesang

Elvin Andreas Pedersen

3.40.2 Member Function Documentation

3.40.2.1 Convert()

```
object?  JobStatusToColorConverter.Convert (
            object?  value,
            Type targetType,
            object?  parameter,
            CultureInfo culture )  [inline]
```

Converts a job status to a color.

Parameters

value	The JobStatus to get the color for
targetType	unused
parameter	unused
culture	unused

Returns

An object of type Color representing the job status

Exceptions

ArgumentException	Thrown if the parameter "value" is not of type JobStatus

3.40.2.2 ConvertBack()

```
object?  JobStatusToColorConverter.ConvertBack (
             object?  value,
             Type targetType,
             object?  parameter,
             CultureInfo culture )  [inline]
```

Not implemented.

The documentation for this class was generated from the following file:

- Utilities/Converters/JobStatusToColorConverter.cs

3.41 JobTask Class Reference

The model holding all the job data.

Properties

- string? Name `[get, set]`
  Gets or sets the name of the job.
- ProductionTimes? ProductionTimes `[get, set]`
  Gets or sets the production times for the job ProductionTimes.
- Customer? Customer `[get, set]`
  Gets or sets the customer produced for Customer.
- ImageSource? JobPreview `[get, set]`
  Gets or sets the image of the job.
- OverruleMaterialCutPresets? OverruleMaterialCutPresets `[get, set]`
  Gets or sets the overrule material cut presets for the job OverruleMaterialCutPresets.
- List< IdentificationField >? IdentificationFields `[get, set]`
  Gets or sets the identification fields for the job IdentificationField
- string? Id `[get, set]`
  Gets or sets Id of the job.
- DateTime? CreationDate `[get, set]`
  Gets or sets the creation date of the job.
- DateTime? ModificationDate `[get, set]`
  Gets or sets the modification date of the job.

- DateTime? StartDate `[get, set]`

  Gets or sets the start date of the job.
- DateTime? FinishDate `[get, set]`

  Gets or sets the finish date of the job.
- DateTime? DueDate `[get, set]`

  Gets or sets the due date of the job.
- string? TaskOperator `[get, set]`

  Gets or sets the operator of the job.
- JobStatus? Status `[get, set]`

  Gets or sets the current status of the job JobStatus.
- JobType? Type `[get, set]`

  Gets or sets the type of the job JobType.
- string? StatusInfo `[get, set]`

  Gets or sets additional status information for the job.
- bool? OpenIniPC `[get, set]`

  Gets or sets a value indicating whether or not the job is open in IPC.
- int? AmountProduced `[get, set]`

  Gets or sets the amount produced for the job.
- int? AmountRejected `[get, set]`

  Gets or sets the amount rejected for the job.
- string? JobId `[get, set]`

  Gets or sets the job Id.
- string? JobPartId `[get, set]`

  Gets or sets the job part Id.
- string? ExternalId `[get, set]`

  Gets or sets the external Id used for integration.
- string? DescriptiveName `[get, set]`

  Gets or sets the descriptive name of the job.
- int? JobPriority `[get, set]`

  Gets or sets the job priority.
- string? MaterialName `[get, set]`

  Gets or sets the name of the material used.
- string? MaterialBoardName `[get, set]`

  Gets or sets the board name of the material.
- int? Amount `[get, set]`

  Gets or sets the total amount of copies to be produced.
- double? MediaX `[get, set]`

  Gets or sets the width of the media.
- double? MediaY `[get, set]`

  Gets or sets the height of the media.
- double? MediaThickness `[get, set]`

  Gets or sets the thickness of the media.
- string? MappingPresetName `[get, set]`

  Gets or sets the name of the mapping preset.
- string? CuttingProfileName `[get, set]`

  Gets or sets the name of the cutting profile.
- string? ToolingPresetName `[get, set]`

  Gets or sets the name of the tooling preset.
- string? OptimizationPresetName `[get, set]`

  Gets or sets the name of the optimization preset.
- string? ProductionPresetName `[get, set]`

Gets or sets the name of the production preset.
- string? OriginalCutFileUnc `[get, set]`

  Gets or sets the universal naming convention path to the original cut file.
- string? CutUnc `[get, set]`

  Gets or sets the universal naming convention path to the cut file.
- string? CutPreviewUnc `[get, set]`

  Gets or sets the universal naming convention path to the cut preview file.
- string? PrintPreviewUnc `[get, set]`

  Gets or sets the universal naming convention path to the print preview file.
- string? JdfUnc `[get, set]`

  Gets or sets the universal naming convention path to the JDF file.
- int? SortOrder `[get, set]`

  Gets or sets the sort order of the job in the queue.
- ProductionType? ProductionType `[get, set]`

  Gets or sets the type of production ProductionType.

3.41.1 Detailed Description

The model holding all the job data.

Author

Ole William Skistad Huslende

3.41.2 Property Documentation

3.41.2.1 Amount

```
int? JobTask.Amount [get], [set]
```

Gets or sets the total amount of copies to be produced.

3.41.2.2 AmountProduced

```
int? JobTask.AmountProduced [get], [set]
```

Gets or sets the amount produced for the job.

3.41.2.3 AmountRejected

```
int? JobTask.AmountRejected [get], [set]
```

Gets or sets the amount rejected for the job.

3.41.2.4 CreationDate

```
DateTime? JobTask.CreationDate [get], [set]
```

Gets or sets the creation date of the job.

3.41.2.5 Customer

```
Customer? JobTask.Customer [get], [set]
```

Gets or sets the customer produced for Customer.

3.41.2.6 CutPreviewUnc

```
string? JobTask.CutPreviewUnc [get], [set]
```

Gets or sets the universal naming convention path to the cut preview file.

3.41.2.7 CuttingProfileName

```
string? JobTask.CuttingProfileName [get], [set]
```

Gets or sets the name of the cutting profile.

3.41.2.8 CutUnc

```
string? JobTask.CutUnc [get], [set]
```

Gets or sets the universal naming convention path to the cut file.

3.41.2.9 DescriptiveName

```
string? JobTask.DescriptiveName [get], [set]
```

Gets or sets the descriptive name of the job.

3.41.2.10 DueDate

```
DateTime? JobTask.DueDate [get], [set]
```

Gets or sets the due date of the job.

3.41.2.11 ExternalId

```
string? JobTask.ExternalId [get], [set]
```

Gets or sets the external Id used for integration.

3.41.2.12 FinishDate

```
DateTime? JobTask.FinishDate [get], [set]
```

Gets or sets the finish date of the job.

3.41.2.13 Id

```
string? JobTask.Id [get], [set]
```

Gets or sets Id of the job.

3.41.2.14 IdentificationFields

```
List<IdentificationField>? JobTask.IdentificationFields [get], [set]
```

Gets or sets the identification fields for the job IdentificationField

3.41.2.15 JdfUnc

```
string?  JobTask.JdfUnc  [get], [set]
```

Gets or sets the universal naming convention path to the JDF file.

3.41.2.16 JobId

```
string?  JobTask.JobId  [get], [set]
```

Gets or sets the job Id.

3.41.2.17 JobPartId

```
string?  JobTask.JobPartId  [get], [set]
```

Gets or sets the job part Id.

3.41.2.18 JobPreview

```
ImageSource?  JobTask.JobPreview  [get], [set]
```

Gets or sets the image of the job.

3.41.2.19 JobPriority

```
int?  JobTask.JobPriority  [get], [set]
```

Gets or sets the job priority.

3.41.2.20 MappingPresetName

```
string?  JobTask.MappingPresetName  [get], [set]
```

Gets or sets the name of the mapping preset.

3.41.2.21 MaterialBoardName

```
string?  JobTask.MaterialBoardName  [get], [set]
```

Gets or sets the board name of the material.

3.41.2.22 MaterialName

```
string?  JobTask.MaterialName  [get], [set]
```

Gets or sets the name of the material used.

3.41.2.23 MediaThickness

```
double?  JobTask.MediaThickness  [get], [set]
```

Gets or sets the thickness of the media.

3.41.2.24 MediaX

```
double?  JobTask.MediaX  [get], [set]
```

Gets or sets the width of the media.

3.41.2.25 MediaY

```
double?  JobTask.MediaY  [get], [set]
```

Gets or sets the height of the media.

3.41.2.26 ModificationDate

```
DateTime?  JobTask.ModificationDate  [get], [set]
```

Gets or sets the modification date of the job.

3.41.2.27 Name

```
string? JobTask.Name [get], [set]
```

Gets or sets the name of the job.

3.41.2.28 OpenIniPC

```
bool? JobTask.OpenIniPC [get], [set]
```

Gets or sets a value indicating whether or not the job is open in IPC.

3.41.2.29 OptimizationPresetName

```
string? JobTask.OptimizationPresetName [get], [set]
```

Gets or sets the name of the optimization preset.

3.41.2.30 OriginalCutFileUnc

```
string? JobTask.OriginalCutFileUnc [get], [set]
```

Gets or sets the universal naming convention path to the original cut file.

3.41.2.31 OverruleMaterialCutPresets

```
OverruleMaterialCutPresets? JobTask.OverruleMaterialCutPresets [get], [set]
```

Gets or sets the overrule material cut presets for the job OverruleMaterialCutPresets.

3.41.2.32 PrintPreviewUnc

```
string? JobTask.PrintPreviewUnc [get], [set]
```

Gets or sets the universal naming convention path to the print preview file.

3.41.2.33 ProductionPresetName

```
string?  JobTask.ProductionPresetName  [get], [set]
```

Gets or sets the name of the production preset.

3.41.2.34 ProductionTimes

```
ProductionTimes?  JobTask.ProductionTimes  [get], [set]
```

Gets or sets the production times for the job ProductionTimes.

3.41.2.35 ProductionType

```
ProductionType?  JobTask.ProductionType  [get], [set]
```

Gets or sets the type of production ProductionType.

3.41.2.36 SortOrder

```
int?  JobTask.SortOrder  [get], [set]
```

Gets or sets the sort order of the job in the queue.

3.41.2.37 StartDate

```
DateTime?  JobTask.StartDate  [get], [set]
```

Gets or sets the start date of the job.

3.41.2.38 Status

```
JobStatus?  JobTask.Status  [get], [set]
```

Gets or sets the current status of the job JobStatus.

3.41.2.39 StatusInfo

```
string?  JobTask.StatusInfo  [get], [set]
```

Gets or sets additional status information for the job.

3.41.2.40 TaskOperator

```
string?  JobTask.TaskOperator  [get], [set]
```

Gets or sets the operator of the job.

3.41.2.41 ToolingPresetName

```
string?  JobTask.ToolingPresetName  [get], [set]
```

Gets or sets the name of the tooling preset.

3.41.2.42 Type

```
JobType?  JobTask.Type  [get], [set]
```

Gets or sets the type of the job JobType.

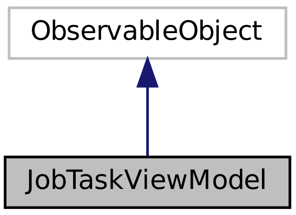
The documentation for this class was generated from the following file:

- Models/ApiQueue/JobTask.cs

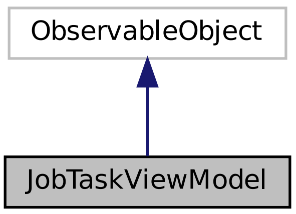
3.42 JobTaskViewModel Class Reference

ViewModel for a job task.

Inheritance diagram for JobTaskViewModel:



Collaboration diagram for JobTaskViewModel:



Public Member Functions

- JobTaskViewModel (IMessenger messenger, JobTask jobTask)

    Constructor for the JobTaskViewModel class.
- void UpdateJobTask ()

    Updates the job task view model with the latest data from the job task model.

3.42.1 Detailed Description

ViewModel for a job task.

This class is used to represent a job task in the UI.

Author

Ole William Skistad Huslende

3.42.2 Constructor & Destructor Documentation

3.42.2.1 JobTaskViewModel()

```
JobTaskViewModel.JobTaskViewModel (
            IMessenger messenger,
            JobTask jobTask ) [inline]
```

Constructor for the JobTaskViewModel class.

Parameters

messenger	The messenger used to send messages between view models inside the table context.
jobTask	The job task model that this view model represents.

3.42.3 Member Function Documentation

3.42.3.1 UpdateJobTask()

```
void JobTaskViewModel.UpdateJobTask ( ) [inline]
```

Updates the job task view model with the latest data from the job task model.

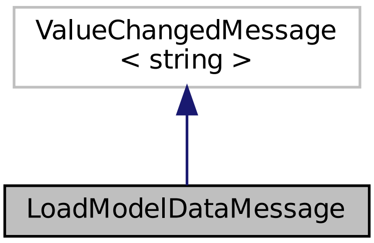
The documentation for this class was generated from the following file:

- ViewModels/TableContextViewModels/SubViewModels/JobTaskViewModel.cs

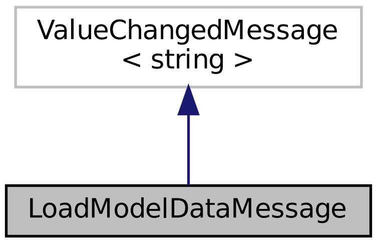
3.43 LoadModelDataMessage Class Reference

Message to load the viewmodels with data after the table model has been loaded.

Inheritance diagram for LoadModelDataMessage:



Collaboration diagram for LoadModelDataMessage:



Public Member Functions

- **LoadModelDataMessage** (string value)

3.43.1 Detailed Description

Message to load the viewmodels with data after the table model has been loaded.

This class is used to send messages between the model and the viewmodel with the help of IMessenger.

Author

Ole William Skistad Huslende

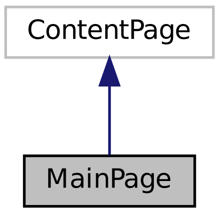
The documentation for this class was generated from the following file:

- Models/ModelMessages.cs

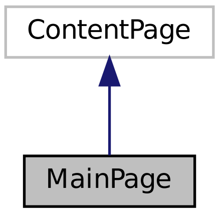
3.44 MainPage Class Reference

MainPage is the main page of the application. It contains the main view model and the tables content view.

Inheritance diagram for MainPage:



Collaboration diagram for MainPage:



Public Member Functions

- MainPage (MainViewModel mainViewModel, TablesContentView tablesContentView)

  Initializes a new instance of the MainPage class.

Protected Member Functions

- override void OnAppearing ()

  Called when the page is appearing. This method is used to reload the tables content view when the page appears.

3.44.1 Detailed Description

MainPage is the main page of the application. It contains the main view model and the tables content view.

Author

Ole William Skistad Huslende

Tormod Smidesang

Elvin Andreas Pedersen

3.44.2 Constructor & Destructor Documentation

3.44.2.1 MainPage()

```
MainPage.MainPage (
            MainViewModel mainViewModel,
            TablesContentView tablesContentView )  [inline]
```

Initializes a new instance of the MainPage class.

Parameters

mainViewModel	Dependency injection of the main view model.
tablesContentView	Dependency injection of the tables content view.

3.44.3 Member Function Documentation

3.44.3.1 OnAppearing()

```
override void MainPage.OnAppearing ( )  [inline], [protected]
```

Called when the page is appearing. This method is used to reload the tables content view when the page appears.

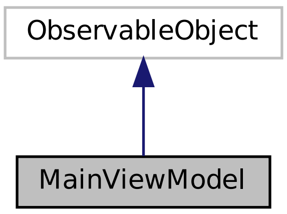
The documentation for this class was generated from the following file:

- Views/MainPage.xaml.cs

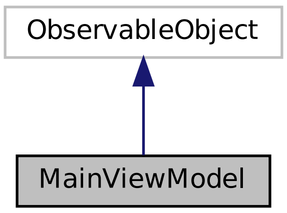
3.45 MainViewModel Class Reference

Main view model for the application.

Inheritance diagram for MainViewModel:



Collaboration diagram for MainViewModel:



Public Member Functions

- MainViewModel (IServiceProvider serviceProvider, TableContextService tableContextService)

  The MainViewModel constructor.
- void TableDragStarting (object sender, DragStartingEventArgs e)

  Handles the drag starting event for a table.
- void TableDrop (object sender, DropEventArgs e)

  Handles the drag ending event for a table.
- async Task GoToTable (TableContext table)

  Navigates to the specified table context.
- void ReconnectTable (TableContext table)

  Reconnects to the specified table context.
- void EditTable (TableContext table)

  Edits the specified table context. By opening a popup.EditTablePopup"/>
- async void DeleteTable (TableContext tableContext)

  Deletes the specified table context. By opening a confirmation dialog.

Properties

- IReadOnlyList< TableContext > Tables `[get]`

  The list of tables managed by the application.
- IAsyncRelayCommand GoToTableCommand `[get]`

  Command to navigate to a table context.
- ICommand ReconnectTableCommand `[get]`

  Command to reconnect to a table.
- ICommand EditTableCommand `[get]`

  Command to edit a table context.
- ICommand DeleteTableCommand `[get]`

  Command to delete a table context.

3.45.1 Detailed Description

Main view model for the application.

This class is responsible for managing the main view of the application. Showing all the tables and their status. Responsible for handling user interactions with the table context object TableContext.

Author

Ole William Skistad Huslende

Tormod Smidesang

Elvin Andreas Pedersen

3.45.2 Constructor & Destructor Documentation

3.45.2.1 MainViewModel()

```
MainViewModel.MainViewModel (
              IServiceProvider serviceProvider,
              TableContextService tableContextService )  [inline]
```

The MainViewModel constructor.

Sets up the commands and initializes the table context service.

Parameters

serviceProvider	The service provider used to resolve dependencies.
tableContextService	The service used to manage table contexts.

3.45.3 Member Function Documentation

3.45.3.1 DeleteTable()

```
async void MainViewModel.DeleteTable (
             TableContext tableContext )  [inline]
```

Deletes the specified table context. By opening a confirmation dialog.

Parameters

tableContext	The specified table context

Author

Tormod Smidesang

3.45.3.2 EditTable()

```
void MainViewModel.EditTable (
             TableContext table )  [inline]
```

Edits the specified table context. By opening a popup.EditTablePopup"/>

Parameters

table	The specified table context

3.45.3.3 GoToTable()

```
async Task MainViewModel.GoToTable (
             TableContext table )  [inline]
```

Navigates to the specified table context.

Parameters

table	The specified table context

Returns

A task that represents the asynchronous operation.

Author

Ole William Skistad Huslende

3.45.3.4 ReconnectTable()

```
void MainViewModel.ReconnectTable (
             TableContext table )  [inline]
```

Reconnects to the specified table context.

Parameters

table	The specified table context

Author

Tormod Smidesange

3.45.3.5 TableDragStarting()

```
void MainViewModel.TableDragStarting (
             object sender,
             DragStartingEventArgs e )  [inline]
```

Handles the drag starting event for a table.

Parameters

sender	The element that is being dragged.
e	The event arguments containing the drag starting information

Author

Elvin Andreas Pedersen

3.45.3.6 TableDrop()

```
void MainViewModel.TableDrop (
             object sender,
             DropEventArgs e )  [inline]
```

Handles the drag ending event for a table.

Parameters

sender	The element that is being dragged.
e	The event arguments containing the drag ending information.

Author

Elvin Andreas Pedersen

3.45.4 Property Documentation

3.45.4.1 DeleteTableCommand

```
ICommand MainViewModel.DeleteTableCommand  [get]
```

Command to delete a table context.

3.45.4.2 EditTableCommand

```
ICommand MainViewModel.EditTableCommand  [get]
```

Command to edit a table context.

3.45.4.3 GoToTableCommand

```
IAsyncRelayCommand MainViewModel.GoToTableCommand  [get]
```

Command to navigate to a table context.

3.45.4.4 ReconnectTableCommand

```
ICommand MainViewModel.ReconnectTableCommand  [get]
```

Command to reconnect to a table.

3.45.4.5 Tables

```
IReadOnlyList<TableContext> MainViewModel.Tables  [get]
```

The list of tables managed by the application.

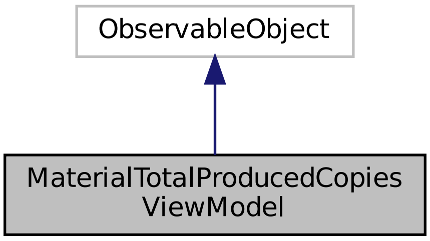
The documentation for this class was generated from the following file:

- ViewModels/MainViewModel.cs

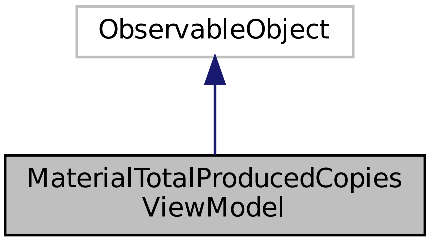
3.46 MaterialTotalProducedCopiesViewModel Class Reference

ViewModel for the material total produced copies.

Inheritance diagram for MaterialTotalProducedCopiesViewModel:



Collaboration diagram for MaterialTotalProducedCopiesViewModel:



3.46.1 Detailed Description

ViewModel for the material total produced copies.

This class is used to represent the total produced copies of a material.

Author

Ole William Skistad Huslende

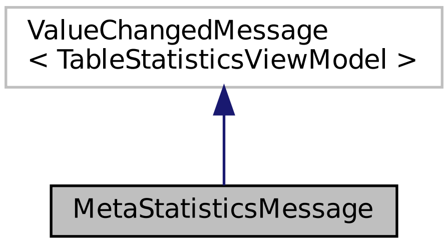
The documentation for this class was generated from the following file:

- ViewModels/TableContextViewModels/SubViewModels/MaterialTotalProducedCopiesViewModel.cs

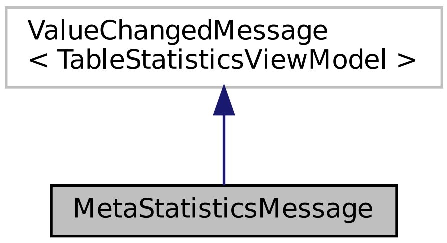
3.47 MetaStatisticsMessage Class Reference

Message sent from the statistics viewmodel to the metastatistics viewmodel when the statistics are updated.

Inheritance diagram for MetaStatisticsMessage:



Collaboration diagram for MetaStatisticsMessage:



Public Member Functions

- **MetaStatisticsMessage** (TableStatisticsViewModel value)

3.47.1 Detailed Description

Message sent from the statistics viewmodel to the metastatistics viewmodel when the statistics are updated.

Author

Ole William Skistad Huslende

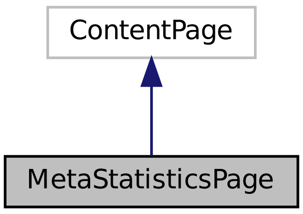
The documentation for this class was generated from the following file:

- ViewModels/ViewModelMessages.cs

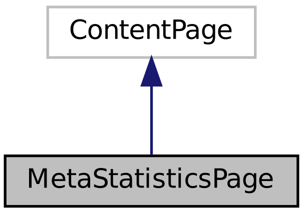
3.48 MetaStatisticsPage Class Reference

MetaStatisticsPage is the page that shows the statistics for all the tables.

Inheritance diagram for MetaStatisticsPage:



Collaboration diagram for MetaStatisticsPage:



Public Member Functions

- MetaStatisticsPage (MetaStatisticsViewModel metaStatisticsViewModel)

    Initializes a new instance of the MetaStatisticsPage class.

3.48.1 Detailed Description

MetaStatisticsPage is the page that shows the statistics for all the tables.

Author

Ole William Skistad Huslende

Elvin Andreas Pedersen

3.48.2 Constructor & Destructor Documentation

3.48.2.1 MetaStatisticsPage()

```
MetaStatisticsPage.MetaStatisticsPage (
            MetaStatisticsViewModel metaStatisticsViewModel )  [inline]
```

Initializes a new instance of the MetaStatisticsPage class.

Parameters

metaStatisticsViewModel	Dependency injection of the view model for the page.

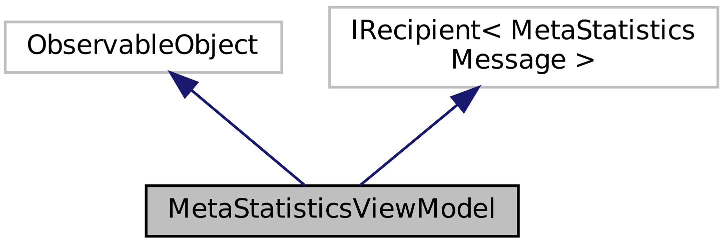
The documentation for this class was generated from the following file:

- Views/MetaStatisticsPage.xaml.cs

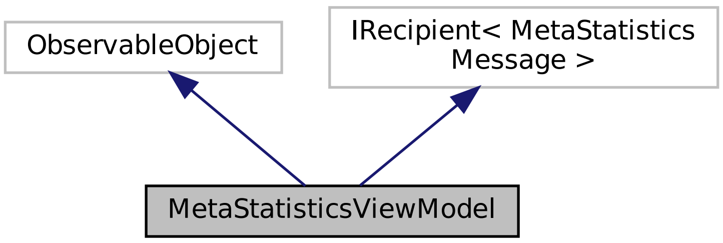
3.49 MetaStatisticsViewModel Class Reference

ViewModel for the MetaStatistics page.

Inheritance diagram for MetaStatisticsViewModel:



Collaboration diagram for MetaStatisticsViewModel:



Public Member Functions

- MetaStatisticsViewModel (IServiceProvider serviceProvider, TableContextService tableContextService, IMessenger metaMessenger)

  Constructor for the MetaStatisticsViewModel.
- void Receive (MetaStatisticsMessage message)

  Handles the reception of a MetaStatisticsMessage.
- ObservableCollection< JobStatisticsViewModel > MergeAndGetTop20 (ObservableCollection< JobStatisticsViewModel > listA, ObservableCollection< JobStatisticsViewModel > listB)

  Merges two collections of job statistics and returns the last 20 based on the start date.
- void GetDayStatistics ()

  Requests the production statistics for the last 24 hours for all tables.
- void GetWeekStatistics ()

  Requests the production statistics for the last 7 days for all tables.
- void GetMonthStatistics ()

  Requests the production statistics for the last 31 days for all tables.
- void RequestProductionStatistics (DateTime before, DateTime after)

  Requests the production statistics for all tables based on the given time range.

3.49.1 Detailed Description

ViewModel for the MetaStatistics page.

Holds the statistics for all the tables connected to the application.

Author

Ole William Skistad Huslende

3.49.2 Constructor & Destructor Documentation

3.49.2.1 MetaStatisticsViewModel()

```
MetaStatisticsViewModel.MetaStatisticsViewModel (
             IServiceProvider serviceProvider,
             TableContextService tableContextService,
             IMessenger metaMessenger )  [inline]
```

Constructor for the MetaStatisticsViewModel.

Defines the columns for the job statistics, material statistics, and customer statistics tables. Registers the view model to receive messages from the messenger.

Parameters

serviceProvider	The service provider for dependency injection.
tableContextService	The table context service for managing table contexts.
metaMessenger	The messenger for sending and receiving messages.

3.49.3 Member Function Documentation

3.49.3.1 GetDayStatistics()

```
void MetaStatisticsViewModel.GetDayStatistics ( )  [inline]
```

Requests the production statistics for the last 24 hours for all tables.

3.49.3.2 GetMonthStatistics()

```
void MetaStatisticsViewModel.GetMonthStatistics ( )  [inline]
```

Requests the production statistics for the last 31 days for all tables.

3.49.3.3 GetWeekStatistics()

```
void MetaStatisticsViewModel.GetWeekStatistics ( )  [inline]
```

Requests the production statistics for the last 7 days for all tables.

3.49.3.4 MergeAndGetTop20()

```
ObservableCollection<JobStatisticsViewModel> MetaStatisticsViewModel.MergeAndGetTop20 (
            ObservableCollection< JobStatisticsViewModel > listA,
            ObservableCollection< JobStatisticsViewModel > listB )  [inline]
```

Merges two collections of job statistics and returns the last 20 based on the start date.

Parameters

listA	The first list of job statistics.
listB	The second list of job statistics.

Returns

Returns a new collection of job statistics containing the last 20 based on the start date.

3.49.3.5 Receive()

```
void MetaStatisticsViewModel.Receive (
            MetaStatisticsMessage message )  [inline]
```

Handles the reception of a MetaStatisticsMessage.

Calculates the statistics for the received message and updates the view model properties.

Parameters

message	The message containing the statistics view model for each of the tables.

3.49.3.6 RequestProductionStatistics()

```
void MetaStatisticsViewModel.RequestProductionStatistics (
            DateTime before,
            DateTime after )  [inline]
```

Requests the production statistics for all tables based on the given time range.

Parameters

before	The statistics requested before this time.
after	The statistics requested after this time.

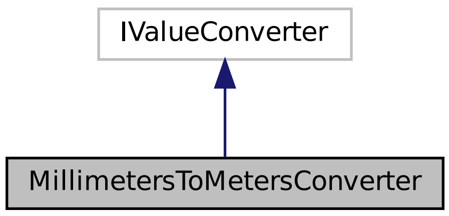
The documentation for this class was generated from the following file:

- ViewModels/MetaStatisticsViewModel.cs

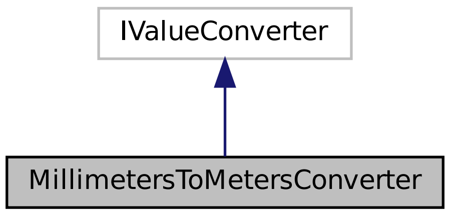
3.50 MillimetersToMetersConverter Class Reference

Converts millimeters to meters.

Inheritance diagram for MillimetersToMetersConverter:



Collaboration diagram for MillimetersToMetersConverter:



Public Member Functions

- object Convert (object value, Type targetType, object parameter, CultureInfo culture)

  Converts an integer representing a distance in millimeters into an equivalent distance expressed in meters.
- object ConvertBack (object value, Type targetType, object parameter, CultureInfo culture)

  Not implemented.

3.50.1 Detailed Description

Converts millimeters to meters.

Author

Elvin Andreas Pedersen

3.50.2 Member Function Documentation

3.50.2.1 Convert()

```
object MillimetersToMetersConverter.Convert (
             object value,
             Type targetType,
             object parameter,
             CultureInfo culture )  [inline]
```

Converts an integer representing a distance in millimeters into an equivalent distance expressed in meters.

Parameters

value	The value to be converted
targetType	unused
parameter	unused
culture	unused

Returns

The same value, but in meters, as a string

3.50.2.2 ConvertBack()

```
object MillimetersToMetersConverter.ConvertBack (
             object value,
             Type targetType,
             object parameter,
             CultureInfo culture )  [inline]
```

Not implemented.

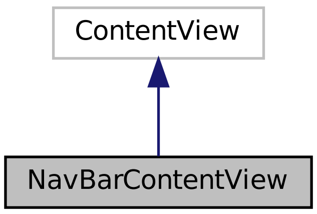
The documentation for this class was generated from the following file:

- Utilities/Converters/MilliMeterToMeterConverter.cs

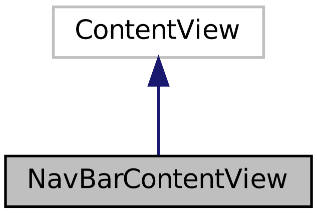
3.51 NavBarContentView Class Reference

Navigation bar that should be put at the top of every page

Inheritance diagram for NavBarContentView:



Collaboration diagram for NavBarContentView:



Public Member Functions

- NavBarContentView ()

  Constructor
- async Task GoToPage (string? pageName)

  Navigates to the provided page

Static Public Attributes

- static readonly BindableProperty IsTablePageProperty

  Bindable bool property set in xaml to determine if the page should show the tabs relevant for all tables (when false) or the tabs relevant for the single selected (when true)
- static readonly BindableProperty PageTitleProperty

  Bindable string property set in xaml. Should contain the name of the page the navbar is displayed on
- static readonly BindableProperty PageSubTitleProperty

  Bindable string property set in xaml. Usage is context dependent, usually shows the name of the relevant table

Properties

- bool IsTablePage `[get, set]`

  See IsTablePageProperty
- string PageTitle `[get, set]`

  See PageTitleProperty
- string PageSubTitle `[get, set]`

  See PageSubTitleProperty
- IAsyncRelayCommand TabClicked `[get]`

  Bindable command called when the user clicks a tab. Must pass the name of the target page as a parameter

3.51.1 Detailed Description

Navigation bar that should be put at the top of every page

This was made as a replacement when we tried to move the home button on the active/history pages so it was in the same place on the tabbar on all pages, but that caused issues with navigation. Making a replacement enabled us to have page name and (when relevant) table name on the top bar

Author

Tormod Smidesang

3.51.2 Constructor & Destructor Documentation

3.51.2.1 NavBarContentView()

```
NavBarContentView.NavBarContentView ( )  [inline]
```

Constructor

3.51.3 Member Function Documentation

3.51.3.1 GoToPage()

```
async Task NavBarContentView.GoToPage (
             string?  pageName )  [inline]
```

Navigates to the provided page

Parameters

pageName	Name of the page to navigate to

Returns

A task representing an asynchronous operation

Exceptions

ArgumentException	Thrown if no page name is provided

3.51.4 Member Data Documentation

3.51.4.1 IsTablePageProperty

```
readonly BindableProperty NavBarContentView.IsTablePageProperty  [static]
```

Initial value:

```
=
        BindableProperty.Create
            (nameof(IsTablePage),
             typeof(bool),
             typeof(NavBarContentView),
             defaultValue: false)
```

Bindable bool property set in xaml to determine if the page should show the tabs relevant for all tables (when false) or the tabs relevant for the single selected (when true)

3.51.4.2 PageSubTitleProperty

```
readonly BindableProperty NavBarContentView.PageSubTitleProperty  [static]
```

Initial value:

```
=
        BindableProperty.Create
            (nameof(PageSubTitle),
             typeof(string),
             typeof(NavBarContentView),
             defaultValue: string.Empty)
```

Bindable string property set in xaml. Usage is context dependent, usually shows the name of the relevant table

3.51.4.3 PageTitleProperty

```
readonly BindableProperty NavBarContentView.PageTitleProperty  [static]
```

Initial value:

```
=
        BindableProperty.Create
            (nameof(PageTitle),
             typeof(string),
             typeof(NavBarContentView),
             defaultValue: string.Empty)
```

Bindable string property set in xaml. Should contain the name of the page the navbar is displayed on

3.51.5 Property Documentation

3.51.5.1 IsTablePage

`bool NavBarContentView.IsTablePage [get], [set]`

See IsTablePageProperty

3.51.5.2 PageSubTitle

`string NavBarContentView.PageSubTitle [get], [set]`

See PageSubTitleProperty

3.51.5.3 PageTitle

`string NavBarContentView.PageTitle [get], [set]`

See PageTitleProperty

3.51.5.4 TabClicked

`IAsyncRelayCommand NavBarContentView.TabClicked [get]`

Bindable command called when the user clicks a tab. Must pass the name of the target page as a parameter

The documentation for this class was generated from the following file:

- Views/NavBarContentView.xaml.cs

3.52 OverruleMaterialCutPresets Class Reference

The model holding overrule material cut preset.

Properties

- bool FocusOnTableTop `[get, set]`
    Force camera to focus on table top, not material
- bool UnconditionalExtraToolLift `[get, set]`
    Force tool to max z height when leaving material.

3.52.1 Detailed Description

The model holding overrule material cut preset.

Author

Ole William Skistad Huslende

3.52.2 Property Documentation

3.52.2.1 FocusOnTableTop

```
bool OverruleMaterialCutPresets.FocusOnTableTop [get], [set]
```

Force camera to focus on table top, not material

3.52.2.2 UnconditionalExtraToolLift

```
bool OverruleMaterialCutPresets.UnconditionalExtraToolLift [get], [set]
```

Force tool to max z height when leaving material.

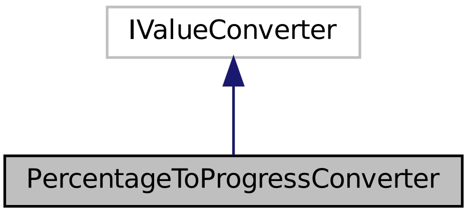
The documentation for this class was generated from the following file:

- Models/ApiQueue/OverruleMaterialCutPresets.cs

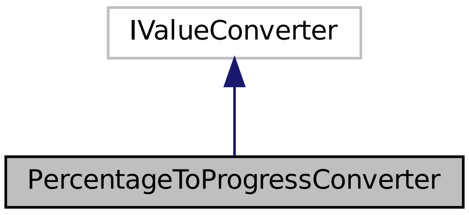
3.53 PercentageToProgressConverter Class Reference

Converts a percentage into a value between 0-1. Can be used the other way as well.

Inheritance diagram for PercentageToProgressConverter:



Collaboration diagram for PercentageToProgressConverter:



Public Member Functions

- object Convert (object value, Type targetType, object parameter, CultureInfo culture)

    Converts a value between 0-100 into a value between 0-1
- object ConvertBack (object value, Type targetType, object parameter, CultureInfo culture)

    Converts a value between 0-1 into a value between 0-100

3.53.1 Detailed Description

Converts a percentage into a value between 0-1. Can be used the other way as well.

Author

Elvin Andreas Pedersen

3.53.2 Member Function Documentation

3.53.2.1 Convert()

```
object PercentageToProgressConverter.Convert (
             object value,
             Type targetType,
             object parameter,
             CultureInfo culture )  [inline]
```

Converts a value between 0-100 into a value between 0-1

Parameters

value	a value between 0-100
targetType	unused
parameter	unused
culture	unused

Returns

a value between 0-1

3.53.2.2 ConvertBack()

```
object PercentageToProgressConverter.ConvertBack (
            object value,
            Type targetType,
            object parameter,
            CultureInfo culture )  [inline]
```

Converts a value between 0-1 into a value between 0-100

Parameters

value	a value between 0-1
targetType	unused
parameter	unused
culture	unused

Returns

a value between 0-100

The documentation for this class was generated from the following file:

- Utilities/Converters/PercentageToProgressConverter.cs

3.54 ProductionTimes Class Reference

Represents a data model for job production times.

Properties

- TimeSpan? TotalProductionTime `[get, set]`

  Gets or sets total time spent producing.
- TimeSpan? EstimatedTimeLastCopy `[get, set]`

  Gets or sets estimated time for the current/latest copy.
- TimeSpan? EstimatedTimeAllCopies `[get, set]`

  Gets or sets estimated time for all copies ordered.
- TimeSpan? RemainingTimeLastCopy `[get, set]`

  Gets or sets remaning time for the current/latest copy.
- TimeSpan? RemainingTimeAllCopies `[get, set]`

  Gets or set remaning time for all remaining ordered copies.
- TimeSpan? ManualHandlingBeforeFirstCopy `[get, set]`

  Gets or set time allocated for manual preperation before each copy.
- TimeSpan? ManualHandlingAfterLastCopy `[get, set]`

  Gets or set time allocated for manual preperation after each copy.

3.54.1 Detailed Description

Represents a data model for job production times.

Author

Ole William Skistad Huslende

3.54.2 Property Documentation

3.54.2.1 EstimatedTimeAllCopies

```
TimeSpan?  ProductionTimes.EstimatedTimeAllCopies  [get], [set]
```

Gets or sets estimated time for all copies ordered.

3.54.2.2 EstimatedTimeLastCopy

```
TimeSpan?  ProductionTimes.EstimatedTimeLastCopy  [get], [set]
```

Gets or sets estimated time for the current/latest copy.

3.54.2.3 ManualHandlingAfterLastCopy

```
TimeSpan?  ProductionTimes.ManualHandlingAfterLastCopy  [get], [set]
```

Gets or set time allocated for manual preperation after each copy.

3.54.2.4 ManualHandlingBeforeFirstCopy

```
TimeSpan?  ProductionTimes.ManualHandlingBeforeFirstCopy  [get], [set]
```

Gets or set time allocated for manual preperation before each copy.

3.54.2.5 RemainingTimeAllCopies

```
TimeSpan?  ProductionTimes.RemainingTimeAllCopies  [get], [set]
```

Gets or set remaning time for all remaining ordered copies.

3.54.2.6 RemainingTimeLastCopy

```
TimeSpan?  ProductionTimes.RemainingTimeLastCopy  [get], [set]
```

Gets or sets remaning time for the current/latest copy.

3.54.2.7 TotalProductionTime

```
TimeSpan?  ProductionTimes.TotalProductionTime  [get], [set]
```

Gets or sets total time spent producing.

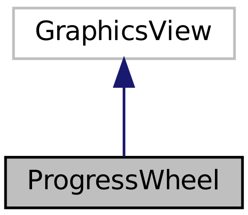
The documentation for this class was generated from the following file:

- Models/ApiQueue/ProductionTimes.cs

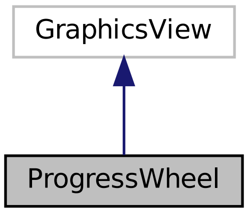
3.55 ProgressWheel Class Reference

A custom progress wheel control for displaying progress in a circular format in the status panel.

Inheritance diagram for ProgressWheel:



Collaboration diagram for ProgressWheel:



Public Member Functions

- ProgressWheel ()

    Constructor for the ProgressWheel class.

Static Public Attributes

- static readonly BindableProperty OuterProgressProperty

    Bindable property for the outer progress value.
- static readonly BindableProperty InnerProgressProperty

    Bindable property for the inner progress value.
- static readonly BindableProperty OuterCircleSizeProperty

    Bindable property for the outer circle size.
- static readonly BindableProperty InnerCircleSizeProperty

    Bindable property for the inner circle size.
- static readonly BindableProperty CircleFillColorProperty

    Bindable property for the circle fill color.
- static readonly BindableProperty CircleBackgroundColorProperty

    Bindable property for the circle background color.

Properties

- float OuterProgress `[get, set]`

    The outer progress value, between 0 and 1.
- float InnerProgress `[get, set]`

    The inner progress value, between 0 and 1.
- float OuterCircleSize `[get, set]`

    The size of the outer circle, between 0 and 1.
- float InnerCircleSize `[get, set]`

    The size of the inner circle, between 0 and 1.
- Color CircleFillColor `[get, set]`

    The color of the circle fill.
- Color CircleBackgroundColor `[get, set]`

    The color of the circle background.

3.55.1 Detailed Description

A custom progress wheel control for displaying progress in a circular format in the status panel.

Author

Ole William Skistad Huslende

3.55.2 Constructor & Destructor Documentation

3.55.2.1 ProgressWheel()

```
ProgressWheel.ProgressWheel ( )  [inline]
```

Constructor for the ProgressWheel class.

3.55.3 Member Data Documentation

3.55.3.1 CircleBackgroundColorProperty

```
readonly BindableProperty ProgressWheel.CircleBackgroundColorProperty  [static]
```

Initial value:

```
=
        BindableProperty.Create(
            nameof(CircleBackgroundColor),
            typeof(Color),
            typeof(ProgressWheel),
            Colors.Gray,
            propertyChanged: OnProgressPropertyChanged)
```

Bindable property for the circle background color.

3.55.3.2 CircleFillColorProperty

```
readonly BindableProperty ProgressWheel.CircleFillColorProperty  [static]
```

Initial value:

```
=
        BindableProperty.Create(
            nameof(CircleFillColor),
            typeof(Color),
            typeof(ProgressWheel),
            Colors.Orange,
            propertyChanged: OnProgressPropertyChanged)
```

Bindable property for the circle fill color.

3.55.3.3 InnerCircleSizeProperty

```
readonly BindableProperty ProgressWheel.InnerCircleSizeProperty  [static]
```

Initial value:

```
=
    BindableProperty.Create(
        nameof(InnerCircleSize),
        typeof(float),
        typeof(ProgressWheel),
        0f,
        propertyChanged: OnProgressPropertyChanged)
```

Bindable property for the inner circle size.

3.55.3.4 InnerProgressProperty

```
readonly BindableProperty ProgressWheel.InnerProgressProperty  [static]
```

Initial value:

```
=
        BindableProperty.Create(
            nameof(InnerProgress),
            typeof(float),
            typeof(ProgressWheel),
            0f,
            propertyChanged: OnProgressPropertyChanged)
```

Bindable property for the inner progress value.

3.55.3.5 OuterCircleSizeProperty

```
readonly BindableProperty ProgressWheel.OuterCircleSizeProperty  [static]
```

Initial value:

```
=
    BindableProperty.Create(
        nameof(OuterCircleSize),
        typeof(float),
        typeof(ProgressWheel),
        0f,
        propertyChanged: OnProgressPropertyChanged)
```

Bindable property for the outer circle size.

3.55.3.6 OuterProgressProperty

```
readonly BindableProperty ProgressWheel.OuterProgressProperty  [static]
```

Initial value:

```
=
        BindableProperty.Create(
            nameof(OuterProgress),
            typeof(float),
            typeof(ProgressWheel),
            0f,
            propertyChanged: OnProgressPropertyChanged)
```

Bindable property for the outer progress value.

3.55.4 Property Documentation

3.55.4.1 CircleBackgroundColor

```
Color ProgressWheel.CircleBackgroundColor  [get], [set]
```

The color of the circle background.

3.55.4.2 CircleFillColor

```
Color ProgressWheel.CircleFillColor  [get], [set]
```

The color of the circle fill.

3.55.4.3 InnerCircleSize

```
float ProgressWheel.InnerCircleSize  [get], [set]
```

The size of the inner circle, between 0 and 1.

3.55.4.4 InnerProgress

```
float ProgressWheel.InnerProgress  [get], [set]
```

The inner progress value, between 0 and 1.

3.55.4.5 OuterCircleSize

```
float ProgressWheel.OuterCircleSize  [get], [set]
```

The size of the outer circle, between 0 and 1.

3.55.4.6 OuterProgress

```
float ProgressWheel.OuterProgress  [get], [set]
```

The outer progress value, between 0 and 1.

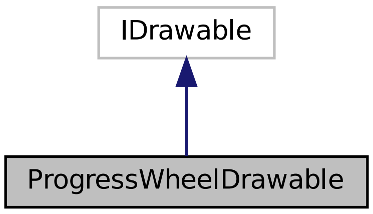
The documentation for this class was generated from the following file:

- Views/Components/ProgressWheel.cs

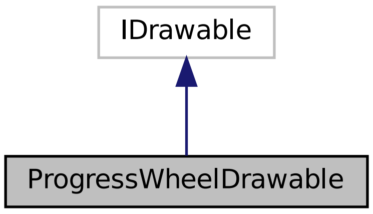
3.56 ProgressWheelDrawable Class Reference

Custom drawable for the ProgressWheel.

Inheritance diagram for ProgressWheelDrawable:



Collaboration diagram for ProgressWheelDrawable:



Public Member Functions

- ProgressWheelDrawable (ProgressWheel wheel)

  Constructor for the ProgressWheelDrawable.
- void Draw (ICanvas canvas, RectF dirtyRect)

  Draws the progress wheel on the canvas.

3.56.1 Detailed Description

Custom drawable for the ProgressWheel.

3.56.2 Constructor & Destructor Documentation

3.56.2.1 ProgressWheelDrawable()

```
ProgressWheelDrawable.ProgressWheelDrawable (
             ProgressWheel wheel )  [inline]
```

Constructor for the ProgressWheelDrawable.

Parameters

wheel	The ProgressWheel instance that this drawable is associated with.

3.56.3 Member Function Documentation

3.56.3.1 Draw()

```
void ProgressWheelDrawable.Draw (
             ICanvas canvas,
             RectF dirtyRect )  [inline]
```

Draws the progress wheel on the canvas.

Parameters

canvas	The canvas to draw on.
dirtyRect	The rectangle that defines the area to draw in.

The documentation for this class was generated from the following file:

- Views/Components/ProgressWheel.cs

3.57 PropertyTrends Class Reference

Represents a data model for property trends sent from the Kongsberg HUB.

Properties

- DateTime FirstOccurrence `[get, set]`

  When the propery first occurred.
- DateTime LastOccurrence `[get, set]`

  When the property last occurred.
- int Occurrences `[get, set]`

  The number of times the property has occurred.
- string LowestValue `[get, set]`

  The lowest value of the property.
- DateTime LowestValueDate `[get, set]`

  When the lowest value occurred.
- string HighestValue `[get, set]`

  The highest value of the property.
- DateTime HighestValueDate `[get, set]`

  When the highest value occurred.
- string Name `[get, set]`

  The name of the property.
- string Value `[get, set]`

  The value of the property.

3.57.1 Detailed Description

Represents a data model for property trends sent from the Kongsberg HUB.

Author

Ole William Skistad Huslende

3.57.2 Property Documentation

3.57.2.1 FirstOccurrence

```
DateTime PropertyTrends.FirstOccurrence  [get], [set]
```

When the propery first occurred.

3.57.2.2 HighestValue

```
string PropertyTrends.HighestValue  [get], [set]
```

The highest value of the property.

3.57.2.3 HighestValueDate

```
DateTime PropertyTrends.HighestValueDate  [get], [set]
```

When the highest value occurred.

3.57.2.4 LastOccurrence

```
DateTime PropertyTrends.LastOccurrence  [get], [set]
```

When the property last occurred.

3.57.2.5 LowestValue

```
string PropertyTrends.LowestValue  [get], [set]
```

The lowest value of the property.

3.57.2.6 LowestValueDate

```
DateTime PropertyTrends.LowestValueDate  [get], [set]
```

When the lowest value occurred.

3.57.2.7 Name

```
string PropertyTrends.Name  [get], [set]
```

The name of the property.

3.57.2.8 Occurrences

```
int PropertyTrends.Occurrences  [get], [set]
```

The number of times the property has occurred.

3.57.2.9 Value

```
string PropertyTrends.Value  [get], [set]
```

The value of the property.

The documentation for this class was generated from the following file:

- Models/ApiTrends/PropertyTrends.cs

3.58 QueueEntryAddedEvent Class Reference

Represents an event from the Kongsberg HUB where a job is added to the queue.

Properties

- JobTask TaskDetails `[get, set]`
  The job that was added to the queue.
- string ID `[get, set]`
  The Id of the event.
- EventAction Action `[get, set]`
  The action that triggered the event EventAction.

3.58.1 Detailed Description

Represents an event from the Kongsberg HUB where a job is added to the queue.

Author

Ole William Skistad Huslende

3.58.2 Property Documentation

3.58.2.1 Action

```
EventAction QueueEntryAddedEvent.Action  [get], [set]
```

The action that triggered the event EventAction.

3.58.2.2 ID

```
string QueueEntryAddedEvent.ID  [get], [set]
```

The Id of the event.

3.58.2.3 TaskDetails

```
JobTask QueueEntryAddedEvent.TaskDetails  [get], [set]
```

The job that was added to the queue.

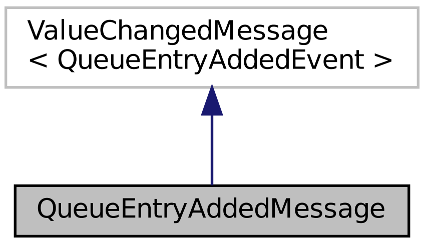
The documentation for this class was generated from the following file:

- Models/ApiSSE/EventTypes.cs

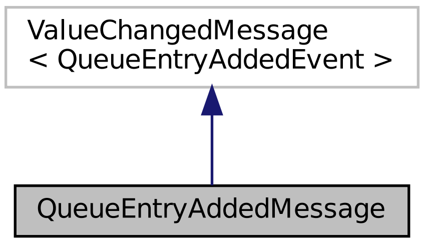
3.59 QueueEntryAddedMessage Class Reference

Message to send queue entry added event received from the Kongsberg HUB and propagate it forward to the viewmodels.

Inheritance diagram for QueueEntryAddedMessage:



Collaboration diagram for QueueEntryAddedMessage:



Public Member Functions

- **QueueEntryAddedMessage** (QueueEntryAddedEvent value)

3.59.1 Detailed Description

Message to send queue entry added event received from the Kongsberg HUB and propagate it forward to the viewmodels.

This class is used to send messages between the model and the viewmodel with the help of IMessenger.

Author

Ole William Skistad Huslende

The documentation for this class was generated from the following file:

- Models/ModelMessages.cs

3.60 QueueEntryMovedEvent Class Reference

Represents an event from the Kongsberg HUB where a job is moved in the queue.

Properties

- int Index `[get, set]`

  The job that was moved in the queue.
- string ID `[get, set]`

  The Id of the event.
- EventAction Action `[get, set]`

  The action that triggered the event EventAction.

3.60.1 Detailed Description

Represents an event from the Kongsberg HUB where a job is moved in the queue.

Author

Ole William Skistad Huslende

3.60.2 Property Documentation

3.60.2.1 Action

```
EventAction QueueEntryMovedEvent.Action  [get], [set]
```

The action that triggered the event EventAction.

3.60.2.2 ID

```
string QueueEntryMovedEvent.ID  [get], [set]
```

The Id of the event.

3.60.2.3 Index

```
int QueueEntryMovedEvent.Index  [get], [set]
```

The job that was moved in the queue.

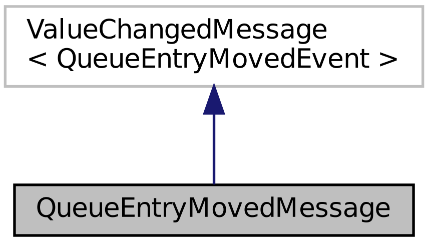
The documentation for this class was generated from the following file:

- Models/ApiSSE/EventTypes.cs

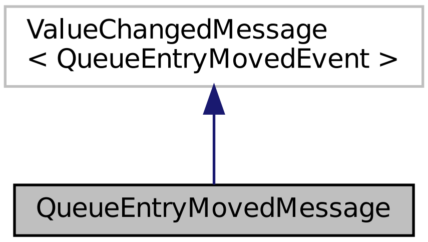
3.61 QueueEntryMovedMessage Class Reference

Message to send queue entry moved event received from the Kongsberg HUB and propagate it forward to the viewmodels.

Inheritance diagram for QueueEntryMovedMessage:



Collaboration diagram for QueueEntryMovedMessage:



Public Member Functions

- **QueueEntryMovedMessage** (QueueEntryMovedEvent value)

3.61.1 Detailed Description

Message to send queue entry moved event received from the Kongsberg HUB and propagate it forward to the viewmodels.

This class is used to send messages between the model and the viewmodel with the help of IMessenger.

Author

Ole William Skistad Huslende

The documentation for this class was generated from the following file:

- Models/ModelMessages.cs

3.62 QueueEntryRemovedEvent Class Reference

Represents an event from the Kongsberg HUB where a job is removed from the queue.

Properties

- string ID `[get, set]`

  The Id of the job that was removed from the queue.
- EventAction Action `[get, set]`

  The action that triggered the event EventAction.

3.62.1 Detailed Description

Represents an event from the Kongsberg HUB where a job is removed from the queue.

Author

Ole William Skistad Huslende

3.62.2 Property Documentation

3.62.2.1 Action

```
EventAction QueueEntryRemovedEvent.Action  [get], [set]
```

The action that triggered the event EventAction.

3.62.2.2 ID

```
string QueueEntryRemovedEvent.ID  [get], [set]
```

The Id of the job that was removed from the queue.

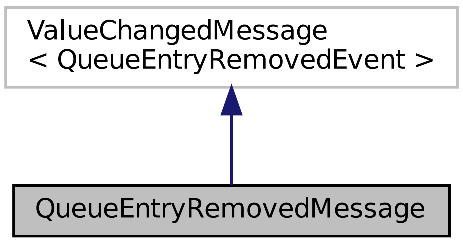
The documentation for this class was generated from the following file:

- Models/ApiSSE/EventTypes.cs

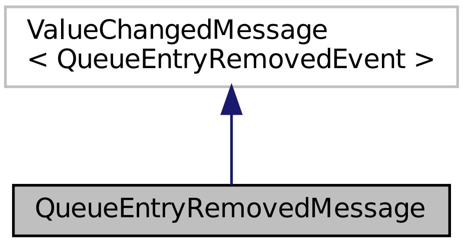
3.63 QueueEntryRemovedMessage Class Reference

Message to send queue entry removed event received from the Kongsberg HUB and propagate it forward to the viewmodels.

Inheritance diagram for QueueEntryRemovedMessage:



Collaboration diagram for QueueEntryRemovedMessage:



Public Member Functions

- **QueueEntryRemovedMessage** (QueueEntryRemovedEvent value)

3.63.1 Detailed Description

Message to send queue entry removed event received from the Kongsberg HUB and propagate it forward to the viewmodels.

This class is used to send messages between the model and the viewmodel with the help of IMessenger.

Author

Ole William Skistad Huslende

The documentation for this class was generated from the following file:

- Models/ModelMessages.cs

3.64 QueueEntryUpdatedEvent Class Reference

Represents an event from the Kongsberg HUB where a job is updated in the queue.

Properties

- JobTask TaskDetails `[get, set]`

  The job properties that have changed.
- string ID `[get, set]`

  The Id of the event.
- EventAction Action `[get, set]`

  The action that triggered the event EventAction.

3.64.1 Detailed Description

Represents an event from the Kongsberg HUB where a job is updated in the queue.

Author

Ole William Skistad Huslende

3.64.2 Property Documentation

3.64.2.1 Action

```
EventAction QueueEntryUpdatedEvent.Action  [get], [set]
```

The action that triggered the event EventAction.

3.64.2.2 ID

```
string QueueEntryUpdatedEvent.ID  [get], [set]
```

The Id of the event.

3.64.2.3 TaskDetails

```
JobTask QueueEntryUpdatedEvent.TaskDetails  [get], [set]
```

The job properties that have changed.

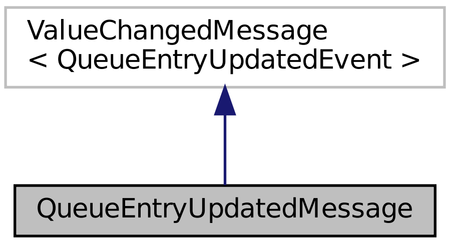
The documentation for this class was generated from the following file:

- Models/ApiSSE/EventTypes.cs

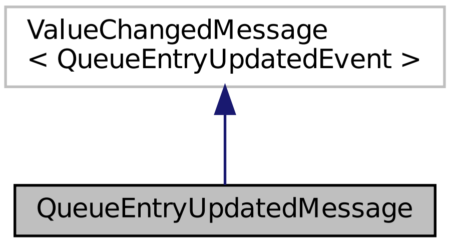
3.65 QueueEntryUpdatedMessage Class Reference

Message to send queue entry updated event received from the Kongsberg HUB and propagate it forward to the viewmodels.

Inheritance diagram for QueueEntryUpdatedMessage:



Collaboration diagram for QueueEntryUpdatedMessage:



Public Member Functions

- **QueueEntryUpdatedMessage** (QueueEntryUpdatedEvent value)

3.65.1 Detailed Description

Message to send queue entry updated event received from the Kongsberg HUB and propagate it forward to the viewmodels.

This class is used to send messages between the model and the viewmodel with the help of IMessenger.

Author

Ole William Skistad Huslende

The documentation for this class was generated from the following file:

- Models/ModelMessages.cs

3.66 QueueServerJobStatistics Class Reference

A data model for the job statistic.

Properties

- string? Id `[get, set]`

    The Id for the job.
- string? Name `[get, set]`

    The name of the job.
- TimeSpan? TotalProductionTime `[get, set]`

    The total production time for the job.
- DateTime? StatisticsCreated `[get, set]`

    When the statistics were created.
- DateTime? StatisticsEnded `[get, set]`

    When the statistics ended.
- int? TotalCopiesProduced `[get, set]`

    Total copies produced for the job.
- int? TotalCopiesRejected `[get, set]`

    Total copies skipped for the job.
- string? **JobId** `[get, set]`
- string? **JobRefInternal** `[get, set]`
- string? **ExternalId** `[get, set]`
- string? **ExternalId2** `[get, set]`
- int? JobPriority `[get, set]`

    The priority of the job.
- DateTime? DueDate `[get, set]`

    The time job is expected to be finished
- Customer? Customer `[get, set]`

    The customer associated with the job ApiQueue.Customer.
- string? OriginalCutFileLoc `[get, set]`

    File location of the original cut file.
- List< QueueServerProductionRunStatistics?>? ProductionRuns `[get, set]`

    List of production runs associated with the job QueueServerProductionRunStatistics.

3.66.1 Detailed Description

A data model for the job statistic.

Author

Ole William Skistad Huslende

3.66.2 Property Documentation

3.66.2.1 Customer

```
Customer? QueueServerJobStatistics.Customer [get], [set]
```

The customer associated with the job ApiQueue.Customer.

3.66.2.2 DueDate

```
DateTime? QueueServerJobStatistics.DueDate [get], [set]
```

The time job is expected to be finished

3.66.2.3 Id

```
string? QueueServerJobStatistics.Id [get], [set]
```

The Id for the job.

3.66.2.4 JobPriority

```
int? QueueServerJobStatistics.JobPriority [get], [set]
```

The priority of the job.

3.66.2.5 Name

```
string? QueueServerJobStatistics.Name [get], [set]
```

The name of the job.

3.66.2.6 OriginalCutFileLoc

```
string? QueueServerJobStatistics.OriginalCutFileLoc [get], [set]
```

File location of the original cut file.

3.66.2.7 ProductionRuns

```
List<QueueServerProductionRunStatistics?>? QueueServerJobStatistics.ProductionRuns [get],
[set]
```

List of production runs associated with the job QueueServerProductionRunStatistics.

3.66.2.8 StatisticsCreated

```
DateTime? QueueServerJobStatistics.StatisticsCreated [get], [set]
```

When the statistics were created.

3.66.2.9 StatisticsEnded

```
DateTime? QueueServerJobStatistics.StatisticsEnded [get], [set]
```

When the statistics ended.

3.66.2.10 TotalCopiesProduced

```
int? QueueServerJobStatistics.TotalCopiesProduced [get], [set]
```

Total copies produced for the job.

3.66.2.11 TotalCopiesRejected

```
int? QueueServerJobStatistics.TotalCopiesRejected [get], [set]
```

Total copies skipped for the job.

3.66.2.12 TotalProductionTime

```
TimeSpan? QueueServerJobStatistics.TotalProductionTime [get], [set]
```

The total production time for the job.

The documentation for this class was generated from the following file:

- Models/ApiStatistics/QueueServerJobStatistics.cs

3.67 QueueServerLayerStatistics Class Reference

A data model for layer statistics in a production run.

Properties

- int? ToolUpDistance `[get, set]`

  The total distance moved by the tool in up position.
- int? ToolUpMoves `[get, set]`

  The total moves made by the tool in up position.
- int? ToolDownDistance `[get, set]`

  The total distance moved by the tool in down position.
- int? ToolDownMoves `[get, set]`

  The total moves made by the tool in down position.
- int? LineCount `[get, set]`

  The total lines produced by the tool.
- int? ArcCount `[get, set]`

  The total arcs produced by the tool.
- string LayerName `[get, set]`

  User defined name of the layer.
- PublicLayerTypes? LayerType `[get, set]`

  The type of the layer PublicLayerTypes.
- string ToolName `[get, set]`

  User defined name of the tool.
- PublicToolTypes? ToolType `[get, set]`

  The type of the tool PublicToolTypes.
- int? Acceleration `[get, set]`

  Acceleration set for layer
- int? SpeedX `[get, set]`

  The X speed set for the layer.
- int? SpeedY `[get, set]`

  The Y speed set for the layer.
- int? DepthAlong `[get, set]`

  Depth along flute direction
- int? DepthAcross `[get, set]`

  Depth across flute direction
- int? NumberOfMultiPasses `[get, set]`

  Number of multi passes set for the layer.
- int? NumberOfRegmarks `[get, set]`

  Number of regmarks in layer
- int? SpindleRpm `[get, set]`

  RPM spindle was ordered to run at.
- int? ToolLag `[get, set]`

  The tool lag set for the layer.
- int? CircleAdjust `[get, set]`

  The tool lag set for the layer.

3.67.1 Detailed Description

A data model for layer statistics in a production run.

Layer statistics are used to provide detailed information about the production run QueueServerProductionRunStatistics.

Author

Ole William Skistad Huslende

3.67.2 Property Documentation

3.67.2.1 Acceleration

```
int? QueueServerLayerStatistics.Acceleration [get], [set]
```

Acceleration set for layer

3.67.2.2 ArcCount

```
int? QueueServerLayerStatistics.ArcCount [get], [set]
```

The total arcs produced by the tool.

3.67.2.3 CircleAdjust

```
int? QueueServerLayerStatistics.CircleAdjust [get], [set]
```

The tool lag set for the layer.

3.67.2.4 DepthAcross

```
int? QueueServerLayerStatistics.DepthAcross [get], [set]
```

Depth across flute direction

3.67.2.5 DepthAlong

```
int?  QueueServerLayerStatistics.DepthAlong  [get], [set]
```

Depth along flute direction

3.67.2.6 LayerName

```
string QueueServerLayerStatistics.LayerName  [get], [set]
```

User defined name of the layer.

3.67.2.7 LayerType

```
PublicLayerTypes?  QueueServerLayerStatistics.LayerType  [get], [set]
```

The type of the layer PublicLayerTypes.

3.67.2.8 LineCount

```
int?  QueueServerLayerStatistics.LineCount  [get], [set]
```

The total lines produced by the tool.

3.67.2.9 NumberOfMultiPasses

```
int?  QueueServerLayerStatistics.NumberOfMultiPasses  [get], [set]
```

Number of multi passes set for the layer.

3.67.2.10 NumberOfRegmarks

```
int?  QueueServerLayerStatistics.NumberOfRegmarks  [get], [set]
```

Number of regmarks in layer

3.67.2.11 SpeedX

```
int? QueueServerLayerStatistics.SpeedX [get], [set]
```

The X speed set for the layer.

3.67.2.12 SpeedY

```
int? QueueServerLayerStatistics.SpeedY [get], [set]
```

The Y speed set for the layer.

3.67.2.13 SpindleRpm

```
int? QueueServerLayerStatistics.SpindleRpm [get], [set]
```

RPM spindle was ordered to run at.

3.67.2.14 ToolDownDistance

```
int? QueueServerLayerStatistics.ToolDownDistance [get], [set]
```

The total distance moved by the tool in down position.

3.67.2.15 ToolDownMoves

```
int? QueueServerLayerStatistics.ToolDownMoves [get], [set]
```

The total moves made by the tool in down position.

3.67.2.16 ToolLag

```
int? QueueServerLayerStatistics.ToolLag [get], [set]
```

The tool lag set for the layer.

3.67.2.17 ToolName

```
string QueueServerLayerStatistics.ToolName  [get], [set]
```

User defined name of the tool.

3.67.2.18 ToolType

```
PublicToolTypes?  QueueServerLayerStatistics.ToolType  [get], [set]
```

The type of the tool PublicToolTypes.

3.67.2.19 ToolUpDistance

```
int?  QueueServerLayerStatistics.ToolUpDistance  [get], [set]
```

The total distance moved by the tool in up position.

3.67.2.20 ToolUpMoves

```
int?  QueueServerLayerStatistics.ToolUpMoves  [get], [set]
```

The total moves made by the tool in up position.

The documentation for this class was generated from the following file:

- Models/ApiStatistics/QueueServerLayerStatistics.cs

3.68 QueueServerProductionRunStatistics Class Reference

A data model for run statistics in a production job.

Properties

- DateTime RunStarted `[get, set]`

  When the production run started.
- DateTime RunEnded `[get, set]`

  When the production run ended.
- int? CopiesOrdered `[get, set]`

  The order number of the copies.
- int? CopiesProduced `[get, set]`

  The number of copies produced.
- int? CopiesRejected `[get, set]`

  The number of copies rejected.
- int? CopiesSkipped `[get, set]`

  The number of copies skipped.
- string Operator `[get, set]`

  Name of signed in Windows user during production run.
- string MappingPresetName `[get, set]`

  Name of the mapping preset used during production run.
- string CuttingProfileName `[get, set]`

  Name of the cutting profile used during production run.
- string ToolingPresetName `[get, set]`

  Name of the tooling preset used during production run.
- string ProductionPresetName `[get, set]`

  Name of the production preset used during production run.
- PublicMeasurementTypes? MaterialMeasurementType `[get, set]`

  The type of material mesurement used during production run PublicMeasurementTypes.
- PublicCompensationTypes? CompensationType `[get, set]`

  The type of compensation used during production run PublicCompensationTypes.
- PublicRegistrationModes? RegistrationMode `[get, set]`

  The type of registration used during production run PublicRegistrationModes.
- ProductionType? ProductionType `[get, set]`

  The type of production used during production run ProductionType.
- Quality? Quality `[get, set]`

  The type of quality used during production run Quality.
- string MaterialName `[get, set]`

  Name of the material used during production run.
- string MaterialFamilyName `[get, set]`

  Name of the material family used during production run.
- int? MaterialThickness `[get, set]`

  The thickness of the material used during production run.
- bool? ExtraToolLiftUsed `[get, set]`

  Is machine instructed to use the extra tool lift during production run.
- int? JobSizeWidth `[get, set]`

  Size of material used during production run.
- string? JobSheetHeight `[get, set]`

  Size of material used during production run.
- int? StepAndRepeatCountX `[get, set]`

  Number of step and repeat jobs in x
- int? StepAndRepeatCountY `[get, set]`

  Number of step and repeat jobs in y
- int? CheckOutDistance `[get, set]`

Distance from corner to check for edge
- bool? SimultaneousCuttingAndUnloading `[get, set]`

  Indicates if stacker is running in simultaneous mode.
- PublicTableCorners? CornerToScanFrom `[get, set]`

  What corner to scan from PublicTableCorners.
- PublicEdgeScanAlongModes? EdgeScanMode `[get, set]`

  The type of edge scan used during production run PublicEdgeScanAlongModes.
- List< CopyTime > CopyTimes `[get, set]`

  Time information for each copy produced during production run.
- List< QueueServerLayerStatistics > Layers `[get, set]`

  List of layers produced during production run.

3.68.1 Detailed Description

A data model for run statistics in a production job.

Run statistics are used to provide detailed information about the production job QueueServerJobStatistics.

Author

Ole William Skistad Huslende

3.68.2 Property Documentation

3.68.2.1 CheckOutDistance

```
int? QueueServerProductionRunStatistics.CheckOutDistance  [get], [set]
```

Distance from corner to check for edge

3.68.2.2 CompensationType

```
PublicCompensationTypes? QueueServerProductionRunStatistics.CompensationType  [get], [set]
```

The type of compensation used during production run PublicCompensationTypes.

3.68.2.3 CopiesOrdered

```
int? QueueServerProductionRunStatistics.CopiesOrdered  [get], [set]
```

The order number of the copies.

3.68.2.4 CopiesProduced

```
int? QueueServerProductionRunStatistics.CopiesProduced [get], [set]
```

The number of copies produced.

3.68.2.5 CopiesRejected

```
int? QueueServerProductionRunStatistics.CopiesRejected [get], [set]
```

The number of copies rejected.

3.68.2.6 CopiesSkipped

```
int? QueueServerProductionRunStatistics.CopiesSkipped [get], [set]
```

The number of copies skipped.

3.68.2.7 CopyTimes

```
List<CopyTime> QueueServerProductionRunStatistics.CopyTimes [get], [set]
```

Time information for each copy produced during production run.

3.68.2.8 CornerToScanFrom

```
PublicTableCorners? QueueServerProductionRunStatistics.CornerToScanFrom [get], [set]
```

What corner to scan from PublicTableCorners.

3.68.2.9 CuttingProfileName

```
string QueueServerProductionRunStatistics.CuttingProfileName [get], [set]
```

Name of the cutting profile used during production run.

3.68.2.10 EdgeScanMode

```
PublicEdgeScanAlongModes? QueueServerProductionRunStatistics.EdgeScanMode [get], [set]
```

The type of edge scan used during production run PublicEdgeScanAlongModes.

3.68.2.11 ExtraToolLiftUsed

```
bool? QueueServerProductionRunStatistics.ExtraToolLiftUsed [get], [set]
```

Is machine instructed to use the extra tool lift during production run.

3.68.2.12 JobSheetHeight

```
string? QueueServerProductionRunStatistics.JobSheetHeight [get], [set]
```

Size of material used during production run.

3.68.2.13 JobSizeWidth

```
int? QueueServerProductionRunStatistics.JobSizeWidth [get], [set]
```

Size of material used during production run.

3.68.2.14 Layers

```
List<QueueServerLayerStatistics> QueueServerProductionRunStatistics.Layers [get], [set]
```

List of layers produced during production run.

3.68.2.15 MappingPresetName

```
string QueueServerProductionRunStatistics.MappingPresetName [get], [set]
```

Name of the mapping preset used during production run.

3.68.2.16 MaterialFamilyName

```
string QueueServerProductionRunStatistics.MaterialFamilyName  [get], [set]
```

Name of the material family used during production run.

3.68.2.17 MaterialMeasurementType

```
PublicMeasurementTypes?  QueueServerProductionRunStatistics.MaterialMeasurementType  [get],
[set]
```

The type of material mesurement used during production run PublicMeasurementTypes.

3.68.2.18 MaterialName

```
string QueueServerProductionRunStatistics.MaterialName  [get], [set]
```

Name of the material used during production run.

3.68.2.19 MaterialThickness

```
int?  QueueServerProductionRunStatistics.MaterialThickness  [get], [set]
```

The thickness of the material used during production run.

3.68.2.20 Operator

```
string QueueServerProductionRunStatistics.Operator  [get], [set]
```

Name of signed in Windows user during production run.

3.68.2.21 ProductionPresetName

```
string QueueServerProductionRunStatistics.ProductionPresetName  [get], [set]
```

Name of the production preset used during production run.

3.68.2.22 ProductionType

```
ProductionType? QueueServerProductionRunStatistics.ProductionType [get], [set]
```

The type of production used during production run ProductionType.

3.68.2.23 Quality

```
Quality? QueueServerProductionRunStatistics.Quality [get], [set]
```

The type of quality used during production run Quality.

3.68.2.24 RegistrationMode

```
PublicRegistrationModes? QueueServerProductionRunStatistics.RegistrationMode [get], [set]
```

The type of registration used during production run PublicRegistrationModes.

3.68.2.25 RunEnded

```
DateTime QueueServerProductionRunStatistics.RunEnded [get], [set]
```

When the production run ended.

3.68.2.26 RunStarted

```
DateTime QueueServerProductionRunStatistics.RunStarted [get], [set]
```

When the production run started.

3.68.2.27 SimultaneousCuttingAndUnloading

```
bool? QueueServerProductionRunStatistics.SimultaneousCuttingAndUnloading [get], [set]
```

Indicates if stacker is running in simultaneous mode.

3.68.2.28 StepAndRepeatCountX

```
int?  QueueServerProductionRunStatistics.StepAndRepeatCountX  [get], [set]
```

Number of step and repeat jobs in x

3.68.2.29 StepAndRepeatCountY

```
int?  QueueServerProductionRunStatistics.StepAndRepeatCountY  [get], [set]
```

Number of step and repeat jobs in y

3.68.2.30 ToolingPresetName

```
string QueueServerProductionRunStatistics.ToolingPresetName  [get], [set]
```

Name of the tooling preset used during production run.

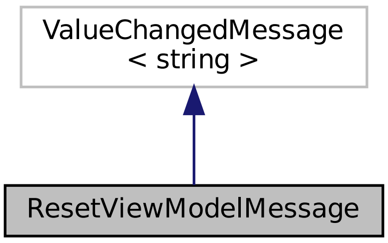
The documentation for this class was generated from the following file:

- Models/ApiStatistics/QueueServerProductionRunStatistics.cs

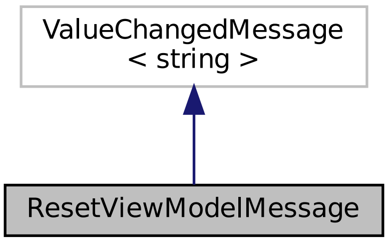
3.69 ResetViewModelMessage Class Reference

Message to reset all the viewmodels inside the table context.

Inheritance diagram for ResetViewModelMessage:



Collaboration diagram for ResetViewModelMessage:



Public Member Functions

- **ResetViewModelMessage** (string value)

3.69.1 Detailed Description

Message to reset all the viewmodels inside the table context.

This class is used to send messages between the model and the viewmodel with the help of IMessenger.

Author

Ole William Skistad Huslende

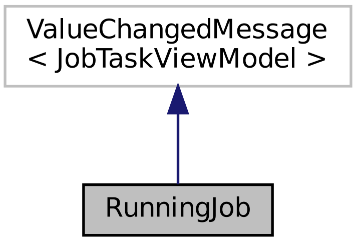
The documentation for this class was generated from the following file:

- Models/ModelMessages.cs

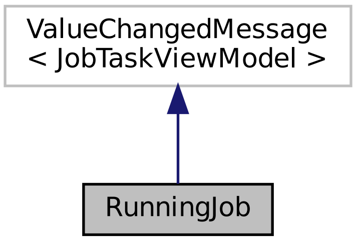
3.70 RunningJob Class Reference

Message sending the current job to the viewmodel

Inheritance diagram for RunningJob:



Collaboration diagram for RunningJob:



Public Member Functions

- **RunningJob** (JobTaskViewModel value)

3.70.1 Detailed Description

Message sending the current job to the viewmodel

Mainly sent from the activejob viewmodel to the other table context viewmodels using the statuspanel.

Author

Ole William Skistad Huslende

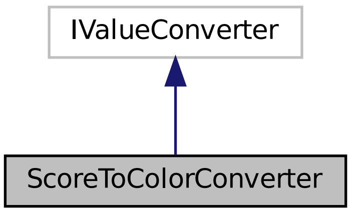
The documentation for this class was generated from the following file:

- ViewModels/ViewModelMessages.cs

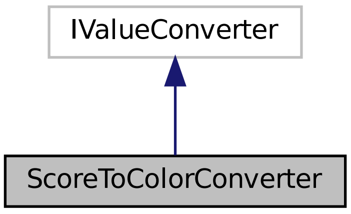
3.71 ScoreToColorConverter Class Reference

Converts a score from 0-100 into a color.

Inheritance diagram for ScoreToColorConverter:



Collaboration diagram for ScoreToColorConverter:



Public Member Functions

- object Convert (object value, Type targetType, object parameter, CultureInfo culture)

  Converts a score from 0-100 into a color.
- object ConvertBack (object value, Type targetType, object parameter, CultureInfo culture)

  Not implemented.

Properties

- Color LowColor = Colors.Red `[get, set]`

  The color for a low score.
- Color MidColor = Colors.Orange `[get, set]`

  The color for a medium score.
- Color HighColor = Colors.Green `[get, set]`

  The color for a low score.

3.71.1 Detailed Description

Converts a score from 0-100 into a color.

Author

Elvin Andreas Pedersen

3.71.2 Member Function Documentation

3.71.2.1 Convert()

```
object ScoreToColorConverter.Convert (
            object value,
            Type targetType,
            object parameter,
            CultureInfo culture )  [inline]
```

Converts a score from 0-100 into a color.

Parameters

value	The score to convert
targetType	unused
parameter	unused
culture	unused

Returns

A color to indicate how good the score is

3.71.2.2 ConvertBack()

```
object ScoreToColorConverter.ConvertBack (
             object value,
             Type targetType,
             object parameter,
             CultureInfo culture )
```

Not implemented.

3.71.3 Property Documentation

3.71.3.1 HighColor

```
Color ScoreToColorConverter.HighColor = Colors.Green  [get], [set]
```

The color for a low score.

3.71.3.2 LowColor

```
Color ScoreToColorConverter.LowColor = Colors.Red  [get], [set]
```

The color for a low score.

3.71.3.3 MidColor

```
Color ScoreToColorConverter.MidColor = Colors.Orange  [get], [set]
```

The color for a medium score.

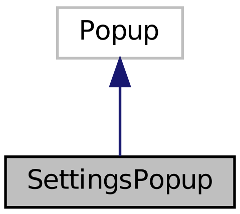
The documentation for this class was generated from the following file:

- Utilities/Converters/ScoreToColorConverter.cs

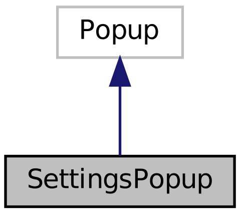
3.72 SettingsPopup Class Reference

Popup for displaying the settings.

Inheritance diagram for SettingsPopup:



Collaboration diagram for SettingsPopup:



Public Member Functions

- SettingsPopup (SettingsViewModel settingsViewModel)
    Initializes a new instance of the SettingsPopup class.

3.72.1 Detailed Description

Popup for displaying the settings.

Author

Tormod Smidesang

Elvin Andreas Pedersen

3.72.2 Constructor & Destructor Documentation

3.72.2.1 SettingsPopup()

```
SettingsPopup.SettingsPopup (
            SettingsViewModel settingsViewModel )  [inline]
```

Initializes a new instance of the SettingsPopup class.

Parameters

settingsViewModel	Dependency injection of the view model for the settings popup.

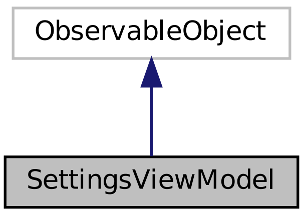
The documentation for this class was generated from the following file:

- Views/Popups/SettingsPopup.xaml.cs

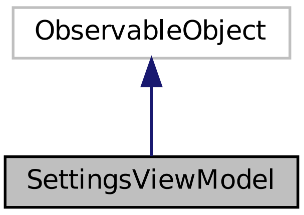
3.73 SettingsViewModel Class Reference

ViewModel for the settings page.

Inheritance diagram for SettingsViewModel:



Collaboration diagram for SettingsViewModel:



Public Member Functions

- SettingsViewModel (UserPreferences userPreferences, TableContextService tableContextService)

    Constructor for the SettingsViewModel.
- async Task DeleteAllCredentials ()

    Command to delete all credentials. Cannot be undone.

Properties

- ObservableCollection< CultureInfo > Languages = [ ] `[get]`

    The list of available languages.

3.73.1 Detailed Description

ViewModel for the settings page.

This class is responsible for managing the settings of the application. Application has to be restarted for the changes to take effect.

Author

Tormod Smidesang

3.73.2 Constructor & Destructor Documentation

3.73.2.1 SettingsViewModel()

```
SettingsViewModel.SettingsViewModel (
            UserPreferences userPreferences,
            TableContextService tableContextService )  [inline]
```

Constructor for the SettingsViewModel.

Parameters

userPreferences	The user preferences service.
tableContextService	The table context service.

3.73.3 Member Function Documentation

3.73.3.1 DeleteAllCredentials()

```
async Task SettingsViewModel.DeleteAllCredentials ( )  [inline]
```

Command to delete all credentials. Cannot be undone.

Returns

Async function

3.73.4 Property Documentation

3.73.4.1 Languages

```
ObservableCollection<CultureInfo> SettingsViewModel.Languages = []  [get]
```

The list of available languages.

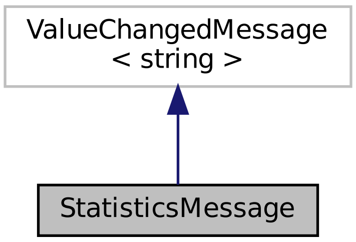
The documentation for this class was generated from the following file:

- ViewModels/Popups/SettingsViewModel.cs

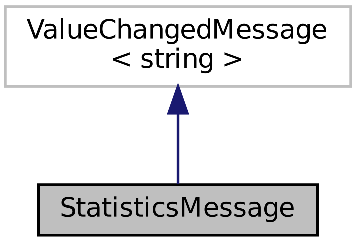
3.74 StatisticsMessage Class Reference

Message to send statistic received from the Kongsberg HUB and propagate it forward to the viewmodels.

Inheritance diagram for StatisticsMessage:



Collaboration diagram for StatisticsMessage:



Public Member Functions

- **StatisticsMessage** (string value)

3.74.1 Detailed Description

Message to send statistic received from the Kongsberg HUB and propagate it forward to the viewmodels.

This class is used to send messages between the model and the viewmodel with the help of IMessenger.

Author

Ole William Skistad Huslende

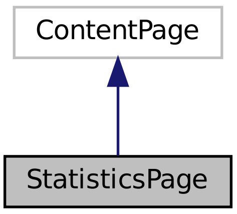
The documentation for this class was generated from the following file:

- Models/ModelMessages.cs

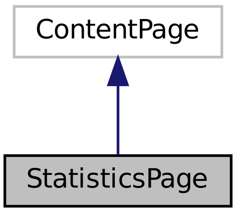
3.75 StatisticsPage Class Reference

StatisticsPage is a page that displays the statistics of a table.

Inheritance diagram for StatisticsPage:



Collaboration diagram for StatisticsPage:



Public Member Functions

- StatisticsPage (TableContextService tableContextService)
  Initializes a new instance of the StatisticsPage class.

Protected Member Functions

- override void OnAppearing ()
  Called when the page is appearing. Sets the binding context to the active table's statistics view model.

3.75.1 Detailed Description

StatisticsPage is a page that displays the statistics of a table.

Author

Ole William Skistad Huslende
Elvin Andreas Pedersen

3.75.2 Constructor & Destructor Documentation

3.75.2.1 StatisticsPage()

```
StatisticsPage.StatisticsPage (
            TableContextService tableContextService )  [inline]
```

Initializes a new instance of the StatisticsPage class.

Parameters

tableContextService	Dependency injection for the table context service.

3.75.3 Member Function Documentation

3.75.3.1 OnAppearing()

```
override void StatisticsPage.OnAppearing ( )  [inline], [protected]
```

Called when the page is appearing. Sets the binding context to the active table's statistics view model.

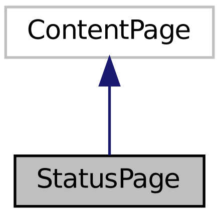
The documentation for this class was generated from the following file:

- Views/TableContextViews/StatisticsPage.xaml.cs

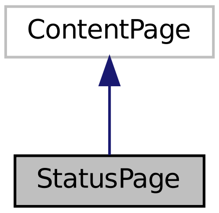
3.76 StatusPage Class Reference

StatusPage is the window that shows the status of the tables.

Inheritance diagram for StatusPage:



Collaboration diagram for StatusPage:



Public Member Functions

- StatusPage (MainViewModel mainViewModel, TablesContentView tablesContentView)

  Initializes a new instance of the StatusPage class.

3.76.1 Detailed Description

StatusPage is the window that shows the status of the tables.

Author

Ole William Skistad Huslende

Elvin Andreas Pedersen

3.76.2 Constructor & Destructor Documentation

3.76.2.1 StatusPage()

```
StatusPage.StatusPage (
            MainViewModel mainViewModel,
            TablesContentView tablesContentView )  [inline]
```

Initializes a new instance of the StatusPage class.

Parameters

mainViewModel	Dependency injection of the main view model.
tablesContentView	Dependency injection of the tables content view.

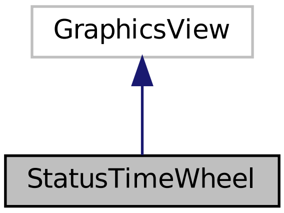
The documentation for this class was generated from the following file:

- Views/StatusPage.xaml.cs

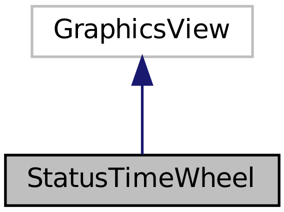
3.77 StatusTimeWheel Class Reference

A custom status time wheel control.

Inheritance diagram for StatusTimeWheel:



Collaboration diagram for StatusTimeWheel:



Public Member Functions

- StatusTimeWheel ()

  Constructor for the StatusTimeWheel class.

Static Public Attributes

- static readonly BindableProperty BusyTimeProperty

  Bindable property for the busy time value.
- static readonly BindableProperty BusyTimeColorProperty

  Bindable property for the busy time color.

- static readonly BindableProperty IdleTimeProperty

  Bindable property for the idle time value.
- static readonly BindableProperty IdleTimeColorProperty

  Bindable property for the idle time color.
- static readonly BindableProperty OffTimeProperty

  Bindable property for the off time value.
- static readonly BindableProperty OffTimeColorProperty

  Bindable property for the off time color.
- static readonly BindableProperty UnknownTimeProperty

  Bindable property for the unknown time value.
- static readonly BindableProperty UnknownTimeColorProperty

  Bindable property for the unknown time color.

Properties

- float BusyTime `[get, set]`

  Gets or sets the busy time value.
- Color BusyTimeColor `[get, set]`

  Gets or sets the busy time color.
- float IdleTime `[get, set]`

  Gets or sets the idle time value.
- Color IdleTimeColor `[get, set]`

  Gets or sets the idle time color.
- float OffTime `[get, set]`

  Gets or sets the off time value.
- Color OffTimeColor `[get, set]`

  Gets or sets the off time color.
- float UnknownTime `[get, set]`

  Gets or sets the unknown time value.
- Color UnknownTimeColor `[get, set]`

  Gets or sets the unknown time color.

3.77.1 Detailed Description

A custom status time wheel control.

Used to display the status of a table in a circular format on the statistics page and the meta statistics page.

Author

Ole William Skistad Huslende

Elvin Andreas Pedersen

3.77.2 Constructor & Destructor Documentation

3.77.2.1 StatusTimeWheel()

```
StatusTimeWheel.StatusTimeWheel ( )  [inline]
```

Constructor for the StatusTimeWheel class.

3.77.3 Member Data Documentation

3.77.3.1 BusyTimeColorProperty

```
readonly BindableProperty StatusTimeWheel.BusyTimeColorProperty  [static]
```

Initial value:

```
=
        BindableProperty.Create(nameof(BusyTimeColor), typeof(Color), typeof(StatusTimeWheel), Colors.Green,
       propertyChanged: OnChanged)
```

Bindable property for the busy time color.

3.77.3.2 BusyTimeProperty

```
readonly BindableProperty StatusTimeWheel.BusyTimeProperty  [static]
```

Initial value:

```
=
        BindableProperty.Create(nameof(BusyTime), typeof(float), typeof(StatusTimeWheel), 0f,
       propertyChanged: OnChanged)
```

Bindable property for the busy time value.

3.77.3.3 IdleTimeColorProperty

```
readonly BindableProperty StatusTimeWheel.IdleTimeColorProperty  [static]
```

Initial value:

```
=
        BindableProperty.Create(nameof(IdleTimeColor), typeof(Color), typeof(StatusTimeWheel),
       Colors.Yellow, propertyChanged: OnChanged)
```

Bindable property for the idle time color.

3.77.3.4 IdleTimeProperty

```
readonly BindableProperty StatusTimeWheel.IdleTimeProperty  [static]
```

Initial value:

```
=
        BindableProperty.Create(nameof(IdleTime), typeof(float), typeof(StatusTimeWheel), 0f,
       propertyChanged: OnChanged)
```

Bindable property for the idle time value.

3.77.3.5 OffTimeColorProperty

```
readonly BindableProperty StatusTimeWheel.OffTimeColorProperty  [static]
```

Initial value:

```
=
        BindableProperty.Create(nameof(OffTimeColor), typeof(Color), typeof(StatusTimeWheel), Colors.Gray,
       propertyChanged: OnChanged)
```

Bindable property for the off time color.

3.77.3.6 OffTimeProperty

```
readonly BindableProperty StatusTimeWheel.OffTimeProperty  [static]
```

Initial value:

```
=
        BindableProperty.Create(nameof(OffTime), typeof(float), typeof(StatusTimeWheel), 0f,
       propertyChanged: OnChanged)
```

Bindable property for the off time value.

3.77.3.7 UnknownTimeColorProperty

```
readonly BindableProperty StatusTimeWheel.UnknownTimeColorProperty  [static]
```

Initial value:

```
=
        BindableProperty.Create(nameof(UnknownTimeColor), typeof(Color), typeof(StatusTimeWheel),
       Colors.Black, propertyChanged: OnChanged)
```

Bindable property for the unknown time color.

3.77.3.8 UnknownTimeProperty

```
readonly BindableProperty StatusTimeWheel.UnknownTimeProperty  [static]
```

Initial value:

```
=
    BindableProperty.Create(nameof(UnknownTime), typeof(float), typeof(StatusTimeWheel), 0f,
       propertyChanged: OnChanged)
```

Bindable property for the unknown time value.

3.77.4 Property Documentation

3.77.4.1 BusyTime

```
float StatusTimeWheel.BusyTime  [get], [set]
```

Gets or sets the busy time value.

3.77.4.2 BusyTimeColor

```
Color StatusTimeWheel.BusyTimeColor  [get], [set]
```

Gets or sets the busy time color.

3.77.4.3 IdleTime

```
float StatusTimeWheel.IdleTime  [get], [set]
```

Gets or sets the idle time value.

3.77.4.4 IdleTimeColor

```
Color StatusTimeWheel.IdleTimeColor  [get], [set]
```

Gets or sets the idle time color.

3.77.4.5 OffTime

```
float StatusTimeWheel.OffTime  [get], [set]
```

Gets or sets the off time value.

3.77.4.6 OffTimeColor

```
Color StatusTimeWheel.OffTimeColor  [get], [set]
```

Gets or sets the off time color.

3.77.4.7 UnknownTime

```
float StatusTimeWheel.UnknownTime  [get], [set]
```

Gets or sets the unknown time value.

3.77.4.8 UnknownTimeColor

```
Color StatusTimeWheel.UnknownTimeColor  [get], [set]
```

Gets or sets the unknown time color.

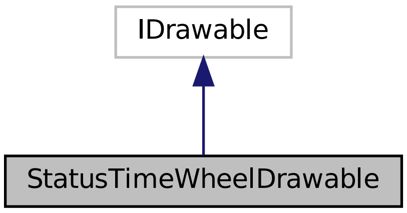
The documentation for this class was generated from the following file:

- Views/Components/StatusTimeWheel.cs

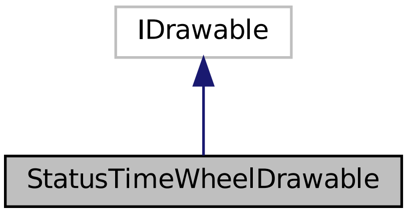
3.78 StatusTimeWheelDrawable Class Reference

Custom drawable for the StatusTimeWheel.

Inheritance diagram for StatusTimeWheelDrawable:



Collaboration diagram for StatusTimeWheelDrawable:



Public Member Functions

- StatusTimeWheelDrawable (StatusTimeWheel wheel)

  Constructor for the StatusTimeWheelDrawable class.
- void Draw (ICanvas canvas, RectF dirtyRect)

  Draws the status time wheel on the canvas.

3.78.1 Detailed Description

Custom drawable for the StatusTimeWheel.

3.78.2 Constructor & Destructor Documentation

3.78.2.1 StatusTimeWheelDrawable()

```
StatusTimeWheelDrawable.StatusTimeWheelDrawable (
             StatusTimeWheel wheel )  [inline]
```

Constructor for the StatusTimeWheelDrawable class.

Parameters

wheel	The StatusTimeWheel instance that this drawable is associated with.

3.78.3 Member Function Documentation

3.78.3.1 Draw()

```
void StatusTimeWheelDrawable.Draw (
             ICanvas canvas,
             RectF dirtyRect )  [inline]
```

Draws the status time wheel on the canvas.

Parameters

canvas	The canvas to draw on.
dirtyRect	The rectangle that defines the area to draw in.

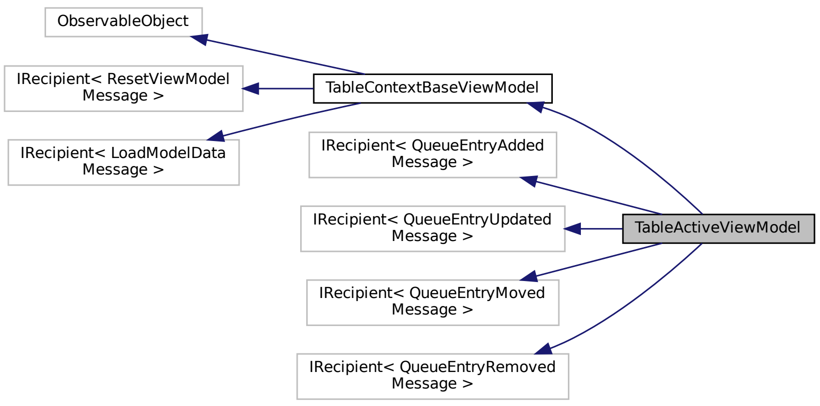
The documentation for this class was generated from the following file:

- Views/Components/StatusTimeWheel.cs

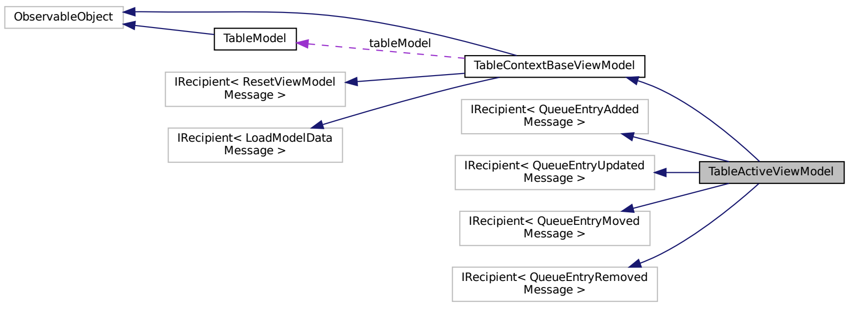
3.79 TableActiveViewModel Class Reference

This class represents the view model for the active jobs.

Inheritance diagram for TableActiveViewModel:


ObservableObject
IRecipient< ResetViewModel Message >
IRecipient< LoadModelData Message >
TableContextBaseViewModel
IRecipient< QueueEntryAdded Message >
IRecipient< QueueEntryUpdated Message >
IRecipient< QueueEntryMoved Message >
IRecipient< QueueEntryRemoved Message >
TableActiveViewModel

Collaboration diagram for TableActiveViewModel:


ObservableObject
TableModel
tableModel
TableContextBaseViewModel
IRecipient< ResetViewModel Message >
IRecipient< LoadModelData Message >
IRecipient< QueueEntryAdded Message >
IRecipient< QueueEntryUpdated Message >
IRecipient< QueueEntryMoved Message >
IRecipient< QueueEntryRemoved Message >
TableActiveViewModel

Public Member Functions

- TableActiveViewModel (IServiceProvider serviceProvider, IMessenger tableContextMessenger, IMessenger metaMessenger, TableModel tableModel)

  The constructor for the TableActiveViewModel class.
- void SortActiveJob (string sortBy)

  Sorts the active jobs in the table by the specified column.
- void ToggleColumnVisibility (string columnName)

  Toggles the visibility of a column in the active jobs table.
- void SetColumnWidth (string columnName, double width)

  Sets the width of a column in the active jobs table.
- void Receive (QueueEntryAddedMessage message)

  Receives a message when a new job is added to the queue.
- void Receive (QueueEntryUpdatedMessage message)

  Receives a message when a job is updated in the queue.
- void Receive (QueueEntryMovedMessage message)

  Receives a message when a job is moved in the queue.

- void Receive (QueueEntryRemovedMessage message)

  Receives a message when a job is removed from the queue.
- void MoveJobToIndex (QueueEntryMovedEvent message)

  Moves a job to a new index in the active jobs collection.
- async Task DeleteSelectedActiveJobAsync ()

  Deletes the selected active job from the table.
- async Task AddActiveJobAsync ()

  Adds a new active job to the table by selecting a ZIP file.

Protected Member Functions

- override void OnResetViewModel ()

  Resets the view model when the table model is reset TableContextBaseViewModel.
- override void OnLoadModelData ()

  Loads the model data when the table model is loaded TableContextBaseViewModel.

Properties

- bool IsJobDetailsPanelVisible `[get]`

  Indicates if the job details panel is visible. If a job is selected, the panel is visible.
- ICommand SortActiveJobsCommand `[get]`

  A command to sort the collumn in the active jobs table.
- IAsyncRelayCommand DeleteSelectedActiveJobAsyncCommand `[get]`

  Command to delete the selected active job from the table.
- IAsyncRelayCommand AddActiveJobAsyncCommand `[get]`

  Command to add a new active job to the table.

Additional Inherited Members

3.79.1 Detailed Description

This class represents the view model for the active jobs.

It contains properties and methods to manage the active jobs in the table. Shows the active jobs in the table and the properties of the jobs.

Author

Ole William Skistad Huslende

Tormod Smidesang

Elvin Andreas Pedersen

3.79.2 Constructor & Destructor Documentation

3.79.2.1 TableActiveViewModel()

```
TableActiveViewModel.TableActiveViewModel (
            IServiceProvider serviceProvider,
            IMessenger tableContextMessenger,
            IMessenger metaMessenger,
            TableModel tableModel )  [inline]
```

The constructor for the TableActiveViewModel class.

Initializes the active jobs collection and registers for messages from the table context TableContext.

Parameters

serviceProvider	The service provider to resolve dependencies.
tableContextMessenger	The messenger to send and receive messages inside the table context TableContext.
metaMessenger	The messenger to send and receive messages outside the table context/>.
tableModel	The model for the table TableModel.

3.79.3 Member Function Documentation

3.79.3.1 AddActiveJobAsync()

```
async Task TableActiveViewModel.AddActiveJobAsync ( )  [inline]
```

Adds a new active job to the table by selecting a ZIP file.

Opens a file picker to select a ZIP file. If the file is valid, it adds the job to the table model TableModel.AddActiveJob(Stream).

Returns

An async operation

Author

Ole William Skistad Huslende

3.79.3.2 DeleteSelectedActiveJobAsync()

```
async Task TableActiveViewModel.DeleteSelectedActiveJobAsync ( )  [inline]
```

Deletes the selected active job from the table.

Returns

An async operation

Author

Ole William Skistad Huslende

3.79.3.3 MoveJobToIndex()

```
void TableActiveViewModel.MoveJobToIndex (
            QueueEntryMovedEvent message )  [inline]
```

Moves a job to a new index in the active jobs collection.

Parameters

message	The message containing the index to move the job to.

Author

Ole William Skistad Huslende

3.79.3.4 OnLoadModelData()

```
override void TableActiveViewModel.OnLoadModelData ( )  [inline], [protected], [virtual]
```

Loads the model data when the table model is loaded TableContextBaseViewModel.

Author

Ole William Skistad Huslende

Reimplemented from TableContextBaseViewModel.

3.79.3.5 OnResetViewModel()

```
override void TableActiveViewModel.OnResetViewModel ( )  [inline], [protected], [virtual]
```

Resets the view model when the table model is reset TableContextBaseViewModel.

Author

Ole William Skistad Huslende

Reimplemented from TableContextBaseViewModel.

3.79.3.6 Receive() [1/4]

```
void TableActiveViewModel.Receive (
             QueueEntryAddedMessage message )  [inline]
```

Receives a message when a new job is added to the queue.

Events are received in the model and propugated to the view model.

Parameters

message	The message containing the details of the new job.

Author

Ole William Skistad Huslende

3.79.3.7 Receive() [2/4]

```
void TableActiveViewModel.Receive (
            QueueEntryMovedMessage message )  [inline]
```

Receives a message when a job is moved in the queue.

Events are received in the model and propugated to the view model.

Parameters

message	

Author

Ole William Skistad Huslende

3.79.3.8 Receive() [3/4]

```
void TableActiveViewModel.Receive (
            QueueEntryRemovedMessage message )  [inline]
```

Receives a message when a job is removed from the queue.

Events are received in the model and propugated to the view model.

Parameters

message	

Author

Ole William Skistad Huslende

3.79.3.9 Receive() [4/4]

```
void TableActiveViewModel.Receive (
            QueueEntryUpdatedMessage message )  [inline]
```

Receives a message when a job is updated in the queue.

Events are received in the model and propugated to the view model.

Parameters

message	The message containing the details of the updated job.

Author

Ole William Skistad Huslende

3.79.3.10 SetColumnWidth()

```
void TableActiveViewModel.SetColumnWidth (
            string columnName,
            double width )  [inline]
```

Sets the width of a column in the active jobs table.

Parameters

columnName	The name of the column to set the width for. The name must match the DisplayName of the column definition.
width	The width to set for the column. The width is in pixels.

Author

Elvin Andreas Pedersen

3.79.3.11 SortActiveJob()

```
void TableActiveViewModel.SortActiveJob (
            string sortBy )  [inline]
```

Sorts the active jobs in the table by the specified column.

Parameters

sortBy	The column to sort by. The column name must match the DisplayName of the column definition.

Author

Ole William Skistad Huslende

3.79.3.12 ToggleColumnVisibility()

```
void TableActiveViewModel.ToggleColumnVisibility (
             string columnName )  [inline]
```

Toggles the visibility of a column in the active jobs table.

Parameters

columnName	The name of the column to toggle visibility for. The name must match the DisplayName of the column definition.

Author

Elvin Andreas Pedersen

3.79.4 Property Documentation

3.79.4.1 AddActiveJobAsyncCommand

```
IAsyncRelayCommand TableActiveViewModel.AddActiveJobAsyncCommand  [get]
```

Command to add a new active job to the table.

3.79.4.2 DeleteSelectedActiveJobAsyncCommand

```
IAsyncRelayCommand TableActiveViewModel.DeleteSelectedActiveJobAsyncCommand  [get]
```

Command to delete the selected active job from the table.

3.79.4.3 IsJobDetailsPanelVisible

```
bool TableActiveViewModel.IsJobDetailsPanelVisible  [get]
```

Indicates if the job details panel is visible. If a job is selected, the panel is visible.

3.79.4.4 SortActiveJobsCommand

```
ICommand TableActiveViewModel.SortActiveJobsCommand  [get]
```

A command to sort the collumn in the active jobs table.

The documentation for this class was generated from the following file:

- ViewModels/TableContextViewModels/TableActiveViewModel.cs

3.80 TableAuthentication Class Reference

Represents a data model for table authentication.

Properties

- string? AccessToken `[get, set]`

  Gets or sets the authentication token.
- DateTime LastUpdated = DateTime.Now `[get]`

  Gets or sets last time authentication token was set.
- bool IsValid = true `[get, set]`

  Gets or sets if the current authentication token is valid.

3.80.1 Detailed Description

Represents a data model for table authentication.

The class contains properties for managing authentication token for the Kongsberg HUB.

Author

Ole William Skistad Huslende

Tormod Smidesang

3.80.2 Property Documentation

3.80.2.1 AccessToken

```
string?  TableAuthentication.AccessToken  [get], [set]
```

Gets or sets the authentication token.

3.80.2.2 IsValid

```
bool TableAuthentication.IsValid = true  [get], [set]
```

Gets or sets if the current authentication token is valid.

3.80.2.3 LastUpdated

```
DateTime TableAuthentication.LastUpdated = DateTime.Now  [get]
```

Gets or sets last time authentication token was set.

The documentation for this class was generated from the following file:

- Models/ApiQueue/TableAuthentication.cs

3.81 TableContext Class Reference

Contains the table model, the view models that read from it and the messenger that tells the viewmodels when to update their information

Public Member Functions

- TableContext (IServiceProvider serviceProvider)

  Constructor. Instantiates the messenger, model and the viewmodels

Properties

- IMessenger TableContextMessenger `[get]`

  Messenger used by the tablemodel to signal the viewmodels to update their information
- TableModel TableModel `[get]`

  Contains the actual table data
- TablePreviewViewModel TablePreviewViewModel `[get]`

  Viewmodel for the elements of the list of tables on the main page
- TableActiveViewModel TableActiveViewModel `[get]`

  Viewmodel for the active jobs page
- TableHistoryViewModel TableHistoryViewModel `[get]`

  Viewmodel for the history jobs page
- TableStatisticsViewModel TableStatisticsViewModel `[get]`

  Viewmodel for the table statistics page

3.81.1 Detailed Description

Contains the table model, the view models that read from it and the messenger that tells the viewmodels when to update their information

Author

Ole William Skistad Huslende

3.81.2 Constructor & Destructor Documentation

3.81.2.1 TableContext()

```
TableContext.TableContext (
            IServiceProvider serviceProvider )  [inline]
```

Constructor. Instantiates the messenger, model and the viewmodels

Parameters

serviceProvider	Reference to the MAUI service provider

3.81.3 Property Documentation

3.81.3.1 TableActiveViewModel

```
TableActiveViewModel TableContext.TableActiveViewModel  [get]
```

Viewmodel for the active jobs page

3.81.3.2 TableContextMessenger

```
IMessenger TableContext.TableContextMessenger  [get]
```

Messenger used by the tablemodel to signal the viewmodels to update their information

3.81.3.3 TableHistoryViewModel

```
TableHistoryViewModel TableContext.TableHistoryViewModel  [get]
```

Viewmodel for the history jobs page

3.81.3.4 TableModel

```
TableModel TableContext.TableModel  [get]
```

Contains the actual table data

3.81.3.5 TablePreviewViewModel

```
TablePreviewViewModel TableContext.TablePreviewViewModel  [get]
```

Viewmodel for the elements of the list of tables on the main page

3.81.3.6 TableStatisticsViewModel

```
TableStatisticsViewModel TableContext.TableStatisticsViewModel  [get]
```

Viewmodel for the table statistics page

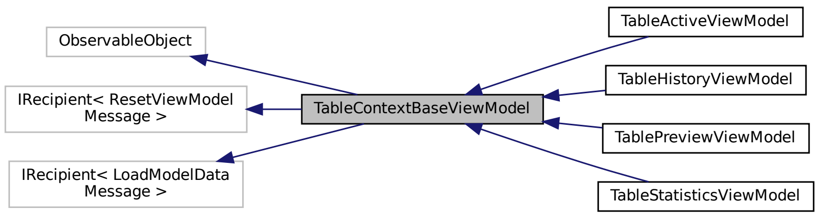
The documentation for this class was generated from the following file:

- TableContext.cs

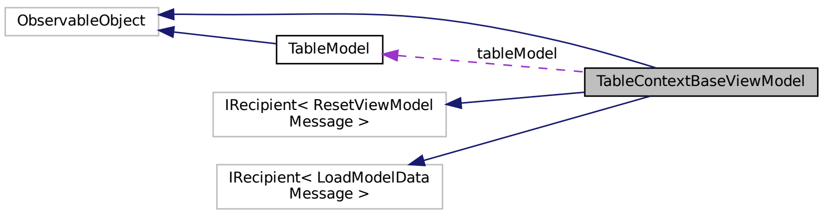
3.82 TableContextBaseViewModel Class Reference

Base class for all table context view models.

Inheritance diagram for TableContextBaseViewModel:



Collaboration diagram for TableContextBaseViewModel:


ObservableObject
TableModel
tableModel
TableContextBaseViewModel
IRecipient< ResetViewModel Message >
IRecipient< LoadModelData Message >

Public Member Functions

- virtual void Receive (ResetViewModelMessage message)

    Resets the view model when a ResetViewModelMessage is received.
- virtual void Receive (LoadModelDataMessage message)

    Loads the model data when a LoadModelDataMessage is received.

Protected Member Functions

- TableContextBaseViewModel (IServiceProvider serviceProvider, IMessenger tableContextMessenger, IMessenger metaMessenger, TableModel tableModel)

    The constructor for the table context base view model.
- virtual void OnResetViewModel ()

    A vertual method to be overridden by derived classes to reset the view model.
- virtual void OnLoadModelData ()

    A virtual method to be overridden by derived classes to load the model data.

Protected Attributes

- readonly IServiceProvider serviceProvider

  The service provider used to resolve dependencies.
- readonly IMessenger tableContextMessenger

  The messenger used to send and receive messages inside the table context TableContext.
- readonly IMessenger metaMessenger

  The messenger used to send and receive messages outside the table context TableContext.
- readonly TableModel tableModel

  The model for the table.

3.82.1 Detailed Description

Base class for all table context view models.

This class implements the IRecipient<T> interface to receive messages from the IMessenger.

Author

Ole William Skistad Huslende

Tormod Smidesang

3.82.2 Constructor & Destructor Documentation

3.82.2.1 TableContextBaseViewModel()

```
TableContextBaseViewModel.TableContextBaseViewModel (
            IServiceProvider serviceProvider,
            IMessenger tableContextMessenger,
            IMessenger metaMessenger,
            TableModel tableModel )  [inline], [protected]
```

The constructor for the table context base view model.

Parameters

serviceProvider	The service provider used to resolve dependencies.
tableContextMessenger	The messenger used to send and receive messages inside the table context TableContext.
metaMessenger	The messenger used to send and receive messages outside the table context TableContext.
tableModel	The model for the table.

3.82.3 Member Function Documentation

3.82.3.1 OnLoadModelData()

```
virtual void TableContextBaseViewModel.OnLoadModelData ( )  [inline], [protected], [virtual]
```

A virtual method to be overridden by derived classes to load the model data.

Author

Ole William Skistad Huslende

Reimplemented in TablePreviewViewModel, TableHistoryViewModel, and TableActiveViewModel.

3.82.3.2 OnResetViewModel()

```
virtual void TableContextBaseViewModel.OnResetViewModel ( )  [inline], [protected], [virtual]
```

A vertual method to be overridden by derived classes to reset the view model.

Author

Ole William Skistad Huslende

Reimplemented in TableStatisticsViewModel, TablePreviewViewModel, TableHistoryViewModel, and TableActiveViewModel.

3.82.3.3 Receive() [1/2]

```
virtual void TableContextBaseViewModel.Receive (
            LoadModelDataMessage message )  [inline], [virtual]
```

Loads the model data when a LoadModelDataMessage is received.

Parameters

message	A message telling the view model to load the model data.

Author

Ole William Skistad Huslende

3.82.3.4 Receive() [2/2]

```
virtual void TableContextBaseViewModel.Receive (
            ResetViewModelMessage message )  [inline], [virtual]
```

Resets the view model when a ResetViewModelMessage is received.

Parameters

message	A message telling the view model to reset itself.

Author

Ole William Skistad Huslende

3.82.4 Member Data Documentation

3.82.4.1 metaMessenger

```
readonly IMessenger TableContextBaseViewModel.metaMessenger  [protected]
```

The messenger used to send and receive messages outside the table context TableContext.

3.82.4.2 serviceProvider

```
readonly IServiceProvider TableContextBaseViewModel.serviceProvider  [protected]
```

The service provider used to resolve dependencies.

3.82.4.3 tableContextMessenger

```
readonly IMessenger TableContextBaseViewModel.tableContextMessenger  [protected]
```

The messenger used to send and receive messages inside the table context TableContext.

3.82.4.4 tableModel

```
readonly TableModel TableContextBaseViewModel.tableModel  [protected]
```

The model for the table.

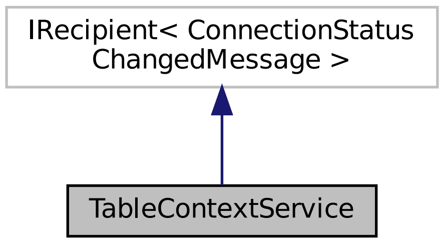
The documentation for this class was generated from the following file:

- ViewModels/TableContextViewModels/TableContextBaseViewModel.cs

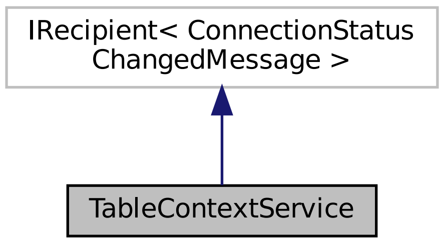
3.83 TableContextService Class Reference

Service responsible for keeping track of the tables

Inheritance diagram for TableContextService:



Collaboration diagram for TableContextService:



Public Member Functions

- TableContextService (IServiceProvider serviceProvider, IMessenger metaMessenger)

  Constructor.
- void HideDisconnectedTables (bool hide)

  Sets the disconnected tables filter to the specified value and updates the filtered list to reflect the change
- void AddTable (TableContext newTable)

  Adds a table to the list. This function calls the function that saves the list of tables to disk
- void RemoveTable (TableContext tableToRemove)

  Removes a table from the list. This function calls the function that saves the list of tables to disk.
- int IndexOfTable (TableContext table)

  Returns the index of the specified table in the list
- void MoveTable (int oldIndex, int newIndex)

Moves the table on the first index to the second index

- void UpdateFilteredList ()

  Updates the filtered list based on filters set by the user
- void Receive (ConnectionStatusChangedMessage message)

  Automatically called when a viewmodel sends the ConnectionStatusChanged message over the metamessenger
- void ClearAllCredentials ()

  Deletes all credentials from memory and disk for every table. Cannot be undone
- async void SaveTablesToDisk ()

  Saves the current list of tables to disk

Properties

- TableContext? ActiveTable `[get, set]`

  The table currently selected by the user (set when user clicks a table on the main page). Used by the active, history and table statistics pages
- IReadOnlyList< TableContext > Tables `[get]`

  A read-only public reference to tables
- IReadOnlyList< TableContext > FilteredTables `[get]`

  A read-only public reference to filteredTables

3.83.1 Detailed Description

Service responsible for keeping track of the tables

Author

Tormod Smidesang

Ole William Skistad Huslende

3.83.2 Constructor & Destructor Documentation

3.83.2.1 TableContextService()

```
TableContextService.TableContextService (
            IServiceProvider serviceProvider,
            IMessenger metaMessenger )  [inline]
```

Constructor.

Parameters

serviceProvider	A reference to the MAUI service provider
metaMessenger	A reference to the metamessenger

3.83.3 Member Function Documentation

3.83.3.1 AddTable()

```
void TableContextService.AddTable (
            TableContext newTable )  [inline]
```

Adds a table to the list. This function calls the function that saves the list of tables to disk

Parameters

newTable	The new table to add to the list

3.83.3.2 ClearAllCredentials()

```
void TableContextService.ClearAllCredentials ( )  [inline]
```

Deletes all credentials from memory and disk for every table. Cannot be undone

3.83.3.3 HideDisconnectedTables()

```
void TableContextService.HideDisconnectedTables (
            bool hide )  [inline]
```

Sets the disconnected tables filter to the specified value and updates the filtered list to reflect the change

Parameters

hide	Will filter out disconnected tables if true, brings them back if false. Calling the function with the same value in this field twice will only update the list

3.83.3.4 IndexOfTable()

```
int TableContextService.IndexOfTable (
            TableContext table )  [inline]
```

Returns the index of the specified table in the list

Parameters

table	The table to get the index of

Returns

The index of the specified table in the list

3.83.3.5 MoveTable()

```
void TableContextService.MoveTable (
            int oldIndex,
            int newIndex )  [inline]
```

Moves the table on the first index to the second index

Parameters

oldIndex	The index of the table to move
newIndex	The new index to move the table to

3.83.3.6 Receive()

```
void TableContextService.Receive (
            ConnectionStatusChangedMessage message )  [inline]
```

Automatically called when a viewmodel sends the ConnectionStatusChanged message over the metamessenger

Parameters

message	Unused, required by IRecipient

3.83.3.7 RemoveTable()

```
void TableContextService.RemoveTable (
            TableContext tableToRemove )  [inline]
```

Removes a table from the list. This function calls the function that saves the list of tables to disk.

Parameters

tableToRemove	A reference to the table to be removed

3.83.3.8 SaveTablesToDisk()

```
async void TableContextService.SaveTablesToDisk ( ) [inline]
```

Saves the current list of tables to disk

3.83.3.9 UpdateFilteredList()

```
void TableContextService.UpdateFilteredList ( ) [inline]
```

Updates the filtered list based on filters set by the user

3.83.4 Property Documentation

3.83.4.1 ActiveTable

```
TableContext? TableContextService.ActiveTable [get], [set]
```

The table currently selected by the user (set when user clicks a table on the main page). Used by the active, history and table statistics pages

3.83.4.2 FilteredTables

```
IReadOnlyList<TableContext> TableContextService.FilteredTables [get]
```

A read-only public reference to filteredTables

3.83.4.3 Tables

```
IReadOnlyList<TableContext> TableContextService.Tables [get]
```

A read-only public reference to tables

The documentation for this class was generated from the following file:

- Services/TableContextService.cs

3.84 TableEntryFieldValidator Class Reference

This class is responsible for validating the fields of a table entry.

Public Member Functions

- string ValidateName (string? TableName)

  Validates the name of the table entry.
- string ValidateIp (string? IpAddress)

  Validates the IP address of the table entry.
- string ValidateClientId (string? ClientId)

  Validates the client ID of the table entry.
- string ValidateClientSecret (string? ClientSecret)

  Validates the client secret of the table entry.

3.84.1 Detailed Description

This class is responsible for validating the fields of a table entry.

Author

Ole William Skistad Huslende

3.84.2 Member Function Documentation

3.84.2.1 ValidateClientId()

```
string TableEntryFieldValidator.ValidateClientId (
             string?  ClientId )  [inline]
```

Validates the client ID of the table entry.

The client ID must not be null or empty and must be a valid GUID.

Parameters

Client↩ Id	The client id to validate

Returns

Returns error message or empty string if no error

3.84.2.2 ValidateClientSecret()

```
string TableEntryFieldValidator.ValidateClientSecret (
            string?  ClientSecret )  [inline]
```

Validates the client secret of the table entry.

The client secret must not be null or empty, must not contain spaces, and must be at least 32 characters long.

Parameters

ClientSecret	The client secret to validate

Returns

Returns error message or empty string if no error

3.84.2.3 ValidateIp()

```
string TableEntryFieldValidator.ValidateIp (
            string?  IpAddress )  [inline]
```

Validates the IP address of the table entry.

The IP address must not be null or empty and must be a valid IP address.

Parameters

IpAddress	The IP address to validate.

Returns

Returns error message or empty string if no error

3.84.2.4 ValidateName()

```
string TableEntryFieldValidator.ValidateName (
            string?  TableName )  [inline]
```

Validates the name of the table entry.

The name must not be null or empty.

Parameters

TableName	The name to validate.

Returns

Returns error message or empty string if no error

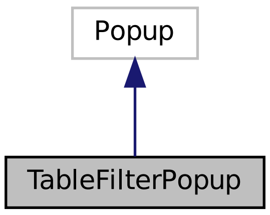
The documentation for this class was generated from the following file:

- Utilities/TableEntryFieldValidator.cs

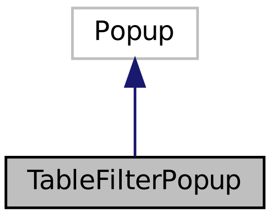
3.85 TableFilterPopup Class Reference

Popup for displaying the filter options for the table list.

Inheritance diagram for TableFilterPopup:



Collaboration diagram for TableFilterPopup:



Public Member Functions

- TableFilterPopup (TableFilterViewModel tableFilterViewModel)

  Initializes a new instance of the TableFilterPopup class.

3.85.1 Detailed Description

Popup for displaying the filter options for the table list.

Author

Tormod Smidesang

Elvin Andreas Pedersen

3.85.2 Constructor & Destructor Documentation

3.85.2.1 TableFilterPopup()

```
TableFilterPopup.TableFilterPopup (
            TableFilterViewModel tableFilterViewModel )  [inline]
```

Initializes a new instance of the TableFilterPopup class.

Parameters

tableFilterViewModel	Dependency injection of the view model for the table filter popup.

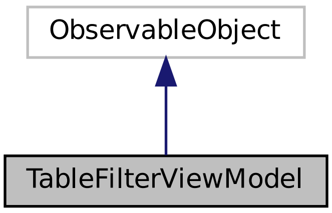
The documentation for this class was generated from the following file:

- Views/Popups/TableFilterPopup.xaml.cs

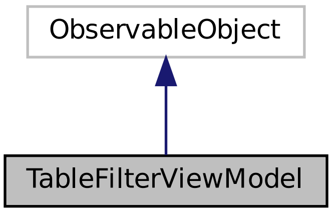
3.86 TableFilterViewModel Class Reference

ViewModel for the table filter popup.

Inheritance diagram for TableFilterViewModel:



Collaboration diagram for TableFilterViewModel:



Public Member Functions

- TableFilterViewModel (TableContextService tableContextService)

  Initializes a new instance of the TableFilterViewModel class.
- void HideDisconnectedTablesChanged (bool hide)

  Sets the hide disconnected tables property and updates the table context service.

3.86.1 Detailed Description

ViewModel for the table filter popup.

Shows a popup with a checkbox to hide disconnected tables. Opens when the user clicks the filter icon in the mainpage.

Author

Tormod Smidesang

3.86.2 Constructor & Destructor Documentation

3.86.2.1 TableFilterViewModel()

```
TableFilterViewModel.TableFilterViewModel (
            TableContextService tableContextService )  [inline]
```

Initializes a new instance of the TableFilterViewModel class.

Parameters

tableContextService	The table context service.

3.86.3 Member Function Documentation

3.86.3.1 HideDisconnectedTablesChanged()

```
void TableFilterViewModel.HideDisconnectedTablesChanged (
             bool hide )  [inline]
```

Sets the hide disconnected tables property and updates the table context service.

Parameters

hide	Whether to hide disconnected tables.

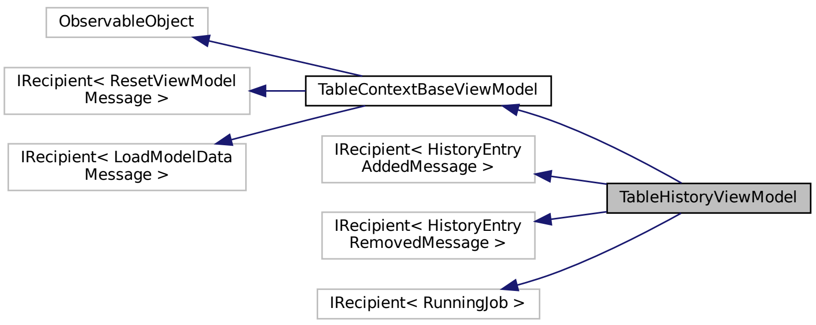
The documentation for this class was generated from the following file:

- ViewModels/Popups/TableFilterViewModel.cs

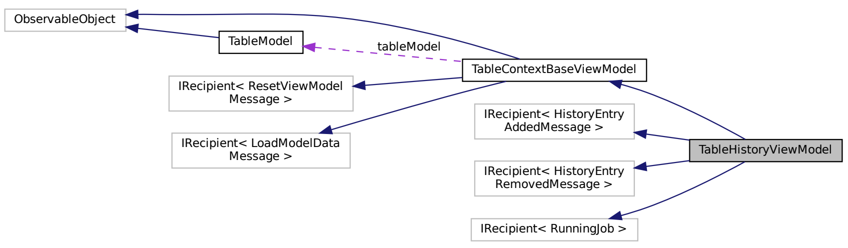
3.87 TableHistoryViewModel Class Reference

This class represents the view model for the history jobs.

Inheritance diagram for TableHistoryViewModel:


ObservableObject
IRecipient< ResetViewModel Message >
IRecipient< LoadModelData Message >
TableContextBaseViewModel
IRecipient< HistoryEntry AddedMessage >
IRecipient< HistoryEntry RemovedMessage >
IRecipient< RunningJob >
TableHistoryViewModel

Collaboration diagram for TableHistoryViewModel:


ObservableObject
TableModel
tableModel
TableContextBaseViewModel
IRecipient< ResetViewModel Message >
IRecipient< LoadModelData Message >
IRecipient< HistoryEntry AddedMessage >
IRecipient< HistoryEntry RemovedMessage >
IRecipient< RunningJob >
TableHistoryViewModel

Public Member Functions

- TableHistoryViewModel (IServiceProvider serviceProvider, IMessenger tableContextMessenger, IMessenger metaMessenger, TableModel tableModel)

    The constructor for the TableHistoryViewModel class.
- void SortHistoryJob (string sortBy)

    Sorts the history jobs in the table by the specified column.
- void ToggleColumnVisibility (string columnName)

    Toggles the visibility of a column in the history jobs table.
- void SetColumnWidth (string columnName, double width)

    Sets the width of a column in the history jobs table.
- void Receive (HistoryEntryAddedMessage message)

    Recives a message when a history entry is added to the table.
- void Receive (RunningJob message)

    Recives a message when the running job is changed.
- void Receive (HistoryEntryRemovedMessage message)

    Recives a message when a history entry is removed from the table.
- async Task DeleteSelectedHistoryJobAsync ()

    Deletes the selected history job from the table.

Protected Member Functions

- override void OnResetViewModel ()

    Resets the view model when the table model is reset TableContextBaseViewModel.
- override void OnLoadModelData ()

    Loads the model data when the table model is loaded TableContextBaseViewModel.

Properties

- ICommand SortHistoryJobsCommand `[get]`

    Command to sort the collection of history jobs.
- bool IsJobDetailsPanelVisible `[get]`

    Indicates if the job details panel is visible.
- IAsyncRelayCommand DeleteSelectedHistoryJobAsyncCommand `[get]`

    Command to delete the selected history job.

Additional Inherited Members

3.87.1 Detailed Description

This class represents the view model for the history jobs.

It contains properties and methods to manage the history jobs in the table. Shows the history jobs in the table and the properties of the jobs.

Author

Ole William Skistad Huslende

Tormod Smidesang

Elvin Andreas Pedersen

3.87.2 Constructor & Destructor Documentation

3.87.2.1 TableHistoryViewModel()

```
TableHistoryViewModel.TableHistoryViewModel (
             IServiceProvider serviceProvider,
             IMessenger tableContextMessenger,
             IMessenger metaMessenger,
             TableModel tableModel )  [inline]
```

The constructor for the TableHistoryViewModel class.

Initializes the history jobs collection and registers for messages from the table context TableContext.

Parameters

serviceProvider	The service provider to resolve dependencies.
tableContextMessenger	The messenger to send and receive messages inside the table context TableContext.
metaMessenger	The messenger to send and receive messages outside the table context/>.
tableModel	The model for the table TableModel.

3.87.3 Member Function Documentation

3.87.3.1 DeleteSelectedHistoryJobAsync()

```
async Task TableHistoryViewModel.DeleteSelectedHistoryJobAsync ( )  [inline]
```

Deletes the selected history job from the table.

<returns An async operation

Author

Ole William Skistad Huslende

3.87.3.2 OnLoadModelData()

```
override void TableHistoryViewModel.OnLoadModelData ( )  [inline], [protected], [virtual]
```

Loads the model data when the table model is loaded TableContextBaseViewModel.

Author

Ole William Skistad Huslende

Reimplemented from TableContextBaseViewModel.

3.87.3.3 OnResetViewModel()

```
override void TableHistoryViewModel.OnResetViewModel ( )  [inline], [protected], [virtual]
```

Resets the view model when the table model is reset TableContextBaseViewModel.

Author

Ole William Skistad Huslende

Reimplemented from TableContextBaseViewModel.

3.87.3.4 Receive() [1/3]

```
void TableHistoryViewModel.Receive (
            HistoryEntryAddedMessage message )  [inline]
```

Recives a message when a history entry is added to the table.

Events are received in the model and propugated to the view model.

Parameters

message	The message contains the job task that was added to the history

Author

Ole William Skistad Huslende

3.87.3.5 Receive() [2/3]

```
void TableHistoryViewModel.Receive (
            HistoryEntryRemovedMessage message )  [inline]
```

Recives a message when a history entry is removed from the table.

Events are received in the model and propugated to the view model.

Parameters

message	The message contains the job task that was removed from the history

Author

Ole William Skistad Huslende

3.87.3.6 Receive() [3/3]

```
void TableHistoryViewModel.Receive (
            RunningJob message )  [inline]
```

Recives a message when the running job is changed.

Parameters

message	The running job

Author

Ole William Skistad Huslende

3.87.3.7 SetColumnWidth()

```
void TableHistoryViewModel.SetColumnWidth (
            string columnName,
            double width )  [inline]
```

Sets the width of a column in the history jobs table.

Parameters

columnName	The name of the column to set the width for. The name must match the DisplayName of the column definition.
width	The width to set for the column. The width is in pixels.

Author

Elvin Andreas Pedersen

3.87.3.8 SortHistoryJob()

```
void TableHistoryViewModel.SortHistoryJob (
            string sortBy )  [inline]
```

Sorts the history jobs in the table by the specified column.

Parameters

sortBy	The column to sort by. The column name must match the DisplayName of the column definition.

Author

Ole William Skistad Huslende

3.87.3.9 ToggleColumnVisibility()

```
void TableHistoryViewModel.ToggleColumnVisibility (
             string columnName )  [inline]
```

Toggles the visibility of a column in the history jobs table.

Parameters

columnName	The name of the column to toggle visibility for. The name must match the DisplayName of the column definition.

Author

Elvin Andreas Pedersen

3.87.4 Property Documentation

3.87.4.1 DeleteSelectedHistoryJobAsyncCommand

```
IAsyncRelayCommand TableHistoryViewModel.DeleteSelectedHistoryJobAsyncCommand  [get]
```

Command to delete the selected history job.

3.87.4.2 IsJobDetailsPanelVisible

```
bool TableHistoryViewModel.IsJobDetailsPanelVisible  [get]
```

Indicates if the job details panel is visible.

3.87.4.3 SortHistoryJobsCommand

```
ICommand TableHistoryViewModel.SortHistoryJobsCommand  [get]
```

Command to sort the collection of history jobs.

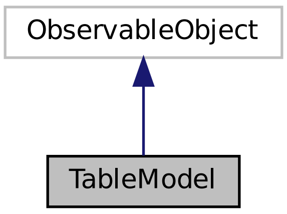
The documentation for this class was generated from the following file:

- ViewModels/TableContextViewModels/TableHistoryViewModel.cs

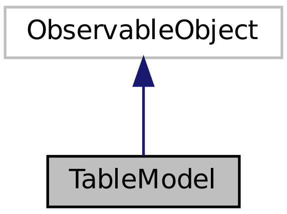
3.88 TableModel Class Reference

The TableModel class represents the model for a Kongsberg table.

Inheritance diagram for TableModel:



Collaboration diagram for TableModel:



Public Member Functions

- TableModel (IServiceProvider serviceProvider, IMessenger messenger)

  Constructor for the TableModel class.
- async void InitializeTable ()

  Initializes the table model.
- void DeleteSavedCredentials ()

  Deletes the saved credentials from disk using secure storage.
- async Task< bool > RequestActiveJobs ()

  Requests the active jobs from the HUB using ApiService.
- async Task< bool > RequestHistoryJobs ()

  Requests the history jobs from the HUB using ApiService.
- async Task RequestJobPreview ()

  Requests the job preview images for all active and history jobs from the HUB using ApiService.
- async Task< bool > RequestTableProperties ()

  Requests the table properties from the HUB using ApiService.
- async Task< bool > RequestTableImage ()

  Requests the image of the table from the HUB using ApiService.
- async Task< bool > DeleteActiveJobTask (string JobId)

  Deletes the active job from the HUB using ApiService.
- async Task< bool > DeleteHistoryJobTask (string JobId)

  Deletes the history job from the HUB using ApiService.
- async Task< bool > SetOrderedCopies (string JobId, int copies)

  Sets the ordered copies for the job on the HUB using ApiService.
- async Task AddActiveJob (Stream stream)

  Adds a new active job to the HUB using ApiService.
- async Task< bool > RequestProductionStatisticsAndStatusTrends (DateTime before, DateTime after)

  Requests the production statistics and status trends from the HUB using ApiService.
- T? TryDeserialize< T > (string json, Action< string >? onError=null)

  Tries to deserialize a JSON string into an object of type T.
- void UpdateJobTask (JobTask updatedJobTask, JobTask incomingData)

  Updates the job task with the new data.
- T? FindProperty< T > (Name name)

  Finds a property by name and converts it to the specified type.

Properties

- required string Name `[get, set]`

  The user defined name of the table.
- required? IPAddress?? **Ip** `[get, set]`
- string? ClientId = string.Empty `[get, set]`

  Client Id for the table.
- string? ClientSecret = string.Empty `[get, set]`

  Client secret for the table.
- JobQueue? ActiveJobs = new() `[get, set]`

  A list of active jobs on the table JobQueue.
- JobQueue? HistoryJobs = new() `[get, set]`

  A list of history jobs on the table JobQueue.
- List< TableProperty >? Properties = [] `[get, set]`

  A list of properties for the table TableProperty.

- ImageSource? TableImage = null `[get, set]`

  The image of the table.
- List< QueueServerJobStatistics?> ProductionStatistics = [ ] `[get, set]`

  A list of production statistics for the table QueueServerJobStatistics.
- List< PropertyTrends?> PropertyTrendsStatus = [ ] `[get, set]`

  A list of property trends for the table PropertyTrends.

3.88.1 Detailed Description

The TableModel class represents the model for a Kongsberg table.

This is the main model for the application where all the data for a specific table is stored.

Author

Ole William Skistad Huslende

Tormod Smidesang

3.88.2 Constructor & Destructor Documentation

3.88.2.1 TableModel()

```
TableModel.TableModel (
            IServiceProvider serviceProvider,
            IMessenger messenger )  [inline]
```

Constructor for the TableModel class.

Sets the table context messenger to send messages to the table specific viewmodels. Dependency injection is used to create the ApiService instance. Starts the event handler for the table.

Parameters

serviceProvider	The service provider to use dependency injection to get instances
messenger	The messenger to send messages to the viewmodels.

3.88.3 Member Function Documentation

3.88.3.1 AddActiveJob()

```
async Task TableModel.AddActiveJob (
            Stream stream )  [inline]
```

Adds a new active job to the HUB using ApiService.

Parameters

stream	The input zip file in stream format

Returns

Task representing an asynchronous operation

Author

Ole William Skistad Huslende

3.88.3.2 DeleteActiveJobTask()

```
async Task<bool> TableModel.DeleteActiveJobTask (
             string JobId )  [inline]
```

Deletes the active job from the HUB using ApiService.

Parameters

Job↩ Id	The Id of the job

Returns

True if request is successful else false

Author

Ole William Skistad Huslende

3.88.3.3 DeleteHistoryJobTask()

```
async Task<bool> TableModel.DeleteHistoryJobTask (
             string JobId )  [inline]
```

Deletes the history job from the HUB using ApiService.

Parameters

Job↩ Id	The Id of the job

Returns

True if request is successful else false

Author

Ole William Skistad Huslende

3.88.3.4 DeleteSavedCredentials()

```
void TableModel.DeleteSavedCredentials ( )  [inline]
```

Deletes the saved credentials from disk using secure storage.

Author

Tormod Smidesang

3.88.3.5 FindProperty< T >()

```
T? TableModel.FindProperty< T > (
             Name name )  [inline]
```

Finds a property by name and converts it to the specified type.

Template Parameters

T	Object to cast into

Parameters

name	Name of the property

Returns

The object

Author

Ole William Skistad Huslende

3.88.3.6 InitializeTable()

```
async void TableModel.InitializeTable ( )  [inline]
```

Initializes the table model.

Sends a message to the messenger to indicate that the table is initializing. Loads the credentials from disk. Sends message to the ViewModel to reset the view. Updates the credentials in the ApiService. Loads data from the HUB Saves the credentials to disk and starts the event listener or stops it if authentication fails.

3.88.3.7 RequestActiveJobs()

```
async Task<bool> TableModel.RequestActiveJobs ( )  [inline]
```

Requests the active jobs from the HUB using ApiService.

Returns

True if request is successful else false

Author

Ole William Skistad Huslende

3.88.3.8 RequestHistoryJobs()

```
async Task<bool> TableModel.RequestHistoryJobs ( )  [inline]
```

Requests the history jobs from the HUB using ApiService.

Returns

True if request is successful else false

Author

Ole William Skistad Huslende

3.88.3.9 RequestJobPreview()

```
async Task TableModel.RequestJobPreview ( )  [inline]
```

Requests the job preview images for all active and history jobs from the HUB using ApiService.

Returns

Async function

Author

Ole William Skistad Huslende

3.88.3.10 RequestProductionStatisticsAndStatusTrends()

```
async Task<bool> TableModel.RequestProductionStatisticsAndStatusTrends (
            DateTime before,
            DateTime after )  [inline]
```

Requests the production statistics and status trends from the HUB using ApiService.

Parameters

before	Before a time
after	After a time

Returns

True if request is successful else false

Author

Ole William Skistad Huslende

3.88.3.11 RequestTableImage()

```
async Task<bool> TableModel.RequestTableImage ( )  [inline]
```

Requests the image of the table from the HUB using ApiService.

Returns

True if request is successful else false

Author

Ole William Skistad Huslende

3.88.3.12 RequestTableProperties()

```
async Task<bool> TableModel.RequestTableProperties ( )  [inline]
```

Requests the table properties from the HUB using ApiService.

Returns

True if request is successful else false

Author

Ole William Skistad Huslende

3.88.3.13 SetOrderedCopies()

```
async Task<bool> TableModel.SetOrderedCopies (
            string JobId,
            int copies )  [inline]
```

Sets the ordered copies for the job on the HUB using ApiService.

Parameters

JobId	The Id of the job
copies	The amount of copies

Returns

True if request is successful else false

Author

Tormod Smidesang

3.88.3.14 TryDeserialize< T >()

```
T? TableModel.TryDeserialize< T > (
            string json,
            Action< string >?  onError = null )  [inline]
```

Tries to deserialize a JSON string into an object of type T.

Template Parameters

T	Object to deserialize into

Parameters

json	JSON string
onError	What to do if error

Returns

Returns the object

Author

Ole William Skistad Huslende

3.88.3.15 UpdateJobTask()

```
void TableModel.UpdateJobTask (
            JobTask updatedJobTask,
            JobTask incomingData )  [inline]
```

Updates the job task with the new data.

Disclaimer: This function was mostly created with assistance from generative AI as the initial function had a bug we were not able to fix ourselves

Parameters

updatedJobTask	The job that needs to be updated
incomingData	The deserialized data

Author

Tormod Smidesang

3.88.4 Property Documentation

3.88.4.1 ActiveJobs

```
JobQueue? TableModel.ActiveJobs = new() [get], [set]
```

A list of active jobs on the table JobQueue.

3.88.4.2 ClientId

```
string? TableModel.ClientId = string.Empty [get], [set]
```

Client Id for the table.

3.88.4.3 ClientSecret

```
string? TableModel.ClientSecret = string.Empty [get], [set]
```

Client secret for the table.

3.88.4.4 HistoryJobs

```
JobQueue? TableModel.HistoryJobs = new() [get], [set]
```

A list of history jobs on the table JobQueue.

3.88.4.5 Name

```
required string TableModel.Name  [get], [set]
```

The user defined name of the table.

3.88.4.6 ProductionStatistics

```
List<QueueServerJobStatistics?> TableModel.ProductionStatistics = []  [get], [set]
```

A list of production statistics for the table QueueServerJobStatistics.

3.88.4.7 Properties

```
List<TableProperty>?  TableModel.Properties = []  [get], [set]
```

A list of properties for the table TableProperty.

3.88.4.8 PropertyTrendsStatus

```
List<PropertyTrends?> TableModel.PropertyTrendsStatus = []  [get], [set]
```

A list of property trends for the table PropertyTrends.

3.88.4.9 TableImage

```
ImageSource?  TableModel.TableImage = null  [get], [set]
```

The image of the table.

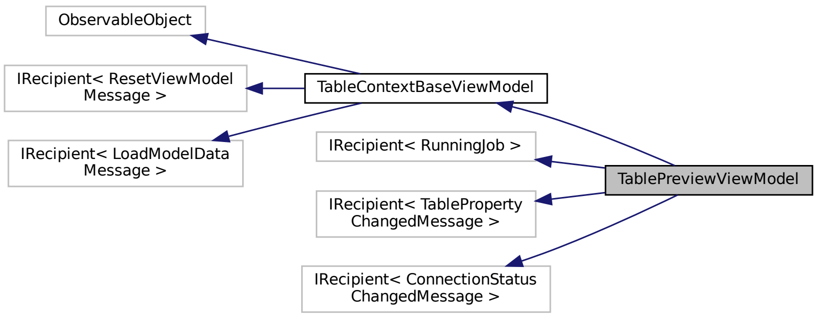
The documentation for this class was generated from the following file:

- Models/TableModel.cs

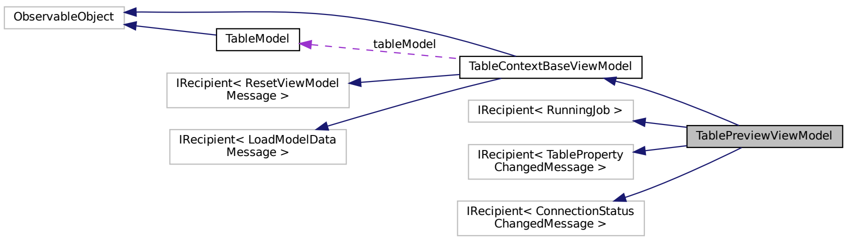
3.89 TablePreviewViewModel Class Reference

ViewModel for the table preview.

Inheritance diagram for TablePreviewViewModel:


ObservableObject
IRecipient< ResetViewModel Message >
TableContextBaseViewModel
IRecipient< LoadModelData Message >
IRecipient< RunningJob >
TablePreviewViewModel
IRecipient< TableProperty ChangedMessage >
IRecipient< ConnectionStatus ChangedMessage >

Collaboration diagram for TablePreviewViewModel:


ObservableObject
TableModel
tableModel
TableContextBaseViewModel
IRecipient< ResetViewModel Message >
IRecipient< LoadModelData Message >
IRecipient< RunningJob >
TablePreviewViewModel
IRecipient< TableProperty ChangedMessage >
IRecipient< ConnectionStatus ChangedMessage >

Public Member Functions

- TablePreviewViewModel (IServiceProvider serviceProvider, IMessenger tableContextMessenger, IMessenger metaMessenger, TableModel tableModel)

  The constructor for the TablePreviewViewModel.
- void Receive (TablePropertyChangedMessage message)

  Receives a message when the table properties are changed.
- void Receive (ConnectionStatusChangedMessage message)

  Receives a message when the connection status is changed.
- void Receive (RunningJob message)

  Receives a message when the running job is changed.
- string GetStatusImagePath ()

  Gets the image path of the table status based on the current status.

Protected Member Functions

- override void OnResetViewModel ()

  Resets the view model to its initial state TableContextBaseViewModel.
- override void OnLoadModelData ()

  Loads the model data from the table model.

Properties

- bool IsConnected `[get]`

  Indicates if the table is connected.
- string FormatedIpcVersion `[get]`

  The IPC version of the table formatted to only show the version number.
- Color TableStatusColor `[get]`

  The color of the table status.
- string TableStatusImagePath `[get]`

  The image path of the table status.

Additional Inherited Members

3.89.1 Detailed Description

ViewModel for the table preview.

It contains properties for the table name, IP address, client ID, client secret, connection status. Used to display the table status and image.

Author

Ole William Skistad Huslende

Tormod Smidesang

3.89.2 Constructor & Destructor Documentation

3.89.2.1 TablePreviewViewModel()

```
TablePreviewViewModel.TablePreviewViewModel (
            IServiceProvider serviceProvider,
            IMessenger tableContextMessenger,
            IMessenger metaMessenger,
            TableModel tableModel )  [inline]
```

The constructor for the TablePreviewViewModel.

Registers the messenger to receive messages from the table context.

Parameters

serviceProvider	The service provider used to resolve dependencies.
tableContextMessenger	The messenger used to send messages between view models inside the table context.
metaMessenger	The messenger used to send messages between view models outside the table context.
tableModel	The table model used to get the table properties.

3.89.3 Member Function Documentation

3.89.3.1 GetStatusImagePath()

```
string TablePreviewViewModel.GetStatusImagePath ( )  [inline]
```

Gets the image path of the table status based on the current status.

Returns

Return the image path of the current status

Author

Ole William Skistad Huslende

3.89.3.2 OnLoadModelData()

```
override void TablePreviewViewModel.OnLoadModelData ( )  [inline], [protected], [virtual]
```

Loads the model data from the table model.

Author

Ole William Skistad Huslende

Reimplemented from TableContextBaseViewModel.

3.89.3.3 OnResetViewModel()

```
override void TablePreviewViewModel.OnResetViewModel ( )  [inline], [protected], [virtual]
```

Resets the view model to its initial state TableContextBaseViewModel.

Author

Ole William Skistad Huslende

Reimplemented from TableContextBaseViewModel.

3.89.3.4 Receive() [1/3]

```
void TablePreviewViewModel.Receive (
             ConnectionStatusChangedMessage message )  [inline]
```

Receives a message when the connection status is changed.

Parameters

message	String containing the new connection status.

Author

Tormod Smidesang

3.89.3.5 Receive() [2/3]

```
void TablePreviewViewModel.Receive (
            RunningJob message )  [inline]
```

Receives a message when the running job is changed.

Parameters

message	The message containing the new running job.

Author

Ole William Skistad Huslende

3.89.3.6 Receive() [3/3]

```
void TablePreviewViewModel.Receive (
            TablePropertyChangedMessage message )  [inline]
```

Receives a message when the table properties are changed.

Events are received in the model and propugated to the view model.

Parameters

message	The message containing the event data.

Author

Ole William Skistad Huslende

3.89.4 Property Documentation

3.89.4.1 FormatedIpcVersion

`string TablePreviewViewModel.FormatedIpcVersion [get]`

The IPC version of the table formatted to only show the version number.

3.89.4.2 IsConnected

`bool TablePreviewViewModel.IsConnected [get]`

Indicates if the table is connected.

3.89.4.3 TableStatusColor

`Color TablePreviewViewModel.TableStatusColor [get]`

The color of the table status.

3.89.4.4 TableStatusImagePath

`string TablePreviewViewModel.TableStatusImagePath [get]`

The image path of the table status.

The documentation for this class was generated from the following file:

- ViewModels/TableContextViewModels/TablePreviewViewModel.cs

3.90 TableProperty Class Reference

Table property is the package each of the properties gets delivered as from the Kongsberg HUB.

Properties

- DataType? DataType `[get, set]`

  Gets or sets the data type of the property Enums.DataType
- DateTime? LastUpdated `[get, set]`

  Gets or sets when the property is last updated.
- int Id `[get, set]`

  Gets or sets Id of the property
- Name? Name `[get, set]`

  Gets or sets the name of the property Enums.Name
- string? Value `[get, set]`

  Gets or sets the value of the property

3.90.1 Detailed Description

Table property is the package each of the properties gets delivered as from the Kongsberg HUB.

Model used for deserialization of the table properties.

Author

Ole William Skistad Huslende

3.90.2 Property Documentation

3.90.2.1 DataType

```
DataType?  TableProperty.DataType  [get], [set]
```

Gets or sets the data type of the property Enums.DataType

3.90.2.2 Id

```
int TableProperty.Id  [get], [set]
```

Gets or sets Id of the property

3.90.2.3 LastUpdated

```
DateTime?  TableProperty.LastUpdated  [get], [set]
```

Gets or sets when the property is last updated.

3.90.2.4 Name

```
Name?  TableProperty.Name  [get], [set]
```

Gets or sets the name of the property Enums.Name

3.90.2.5 Value

```
string? TableProperty.Value [get], [set]
```

Gets or sets the value of the property

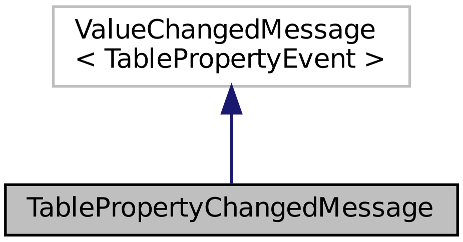
The documentation for this class was generated from the following file:

- Models/ApiProperties/TableProperty.cs

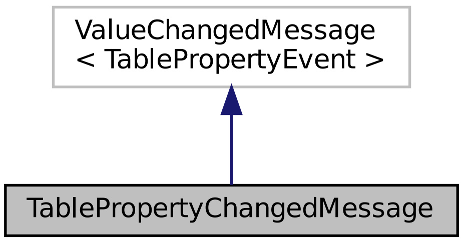
3.91 TablePropertyChangedMessage Class Reference

Message to send table property changed event received from the Kongsberg HUB and propagate it forward to the viewmodels.

Inheritance diagram for TablePropertyChangedMessage:



Collaboration diagram for TablePropertyChangedMessage:



Public Member Functions

- **TablePropertyChangedMessage** (TablePropertyEvent value)

3.91.1 Detailed Description

Message to send table property changed event received from the Kongsberg HUB and propagate it forward to the viewmodels.

This class is used to send messages between the model and the viewmodel with the help of IMessenger.

Author

Ole William Skistad Huslende

The documentation for this class was generated from the following file:

- Models/ModelMessages.cs

3.92 TablePropertyEvent Class Reference

Represents an event from the Kongsberg HUB where one or more properties are changed.

Properties

- List< TableProperty > TableDetails `[get, set]`
  A list of properties that have changed.
- EventAction Action `[get, set]`
  The action that triggered the event EventAction.

3.92.1 Detailed Description

Represents an event from the Kongsberg HUB where one or more properties are changed.

Author

Ole William Skistad Huslende

3.92.2 Property Documentation

3.92.2.1 Action

```
EventAction TablePropertyEvent.Action  [get], [set]
```

The action that triggered the event EventAction.

3.92.2.2 TableDetails

```
List<TableProperty> TablePropertyEvent.TableDetails  [get], [set]
```

A list of properties that have changed.

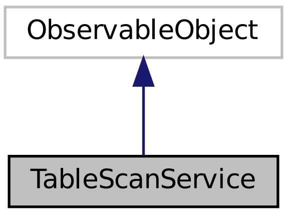
The documentation for this class was generated from the following file:

- Models/ApiSSE/EventTypes.cs

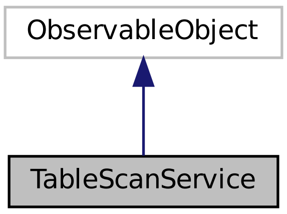
3.93 TableScanService Class Reference

Discovers cutting tables on the local network

Inheritance diagram for TableScanService:



Collaboration diagram for TableScanService:



Public Member Functions

- TableScanService (IServiceProvider serviceProvider)

  Constructor.
- bool Busy ()

  Check if a scan is already in progress.
- async Task Abort ()

  Try to stop a scan if it's running.
- async Task< ConcurrentQueue< TableContext >?> ScanInterface (NetworkInterface networkInterface)

  Scans the provided network interface for cutting tables.
- uint GetNetworkAddress (IPAddress HostAddress, IPAddress SubnetMask)

  Gets the network address of the network the host is on as an unsigned integer.
- uint GetBroadcastAddress (IPAddress HostAddress, IPAddress SubnetMask)

  Gets the broadcast address of the network the host is on as an unsigned integer.

3.93.1 Detailed Description

Discovers cutting tables on the local network

Author

Tormod Smidesang

3.93.2 Constructor & Destructor Documentation

3.93.2.1 TableScanService()

```
TableScanService.TableScanService (
             IServiceProvider serviceProvider )  [inline]
```

Constructor.

Parameters

serviceProvider	Reference to the MAUI service provider

3.93.3 Member Function Documentation

3.93.3.1 Abort()

```
async Task TableScanService.Abort ( )  [inline]
```

Try to stop a scan if it's running.

Returns

A Task object representing an asynchronous operation

3.93.3.2 Busy()

```
bool TableScanService.Busy ( )  [inline]
```

Check if a scan is already in progress.

Returns

true if a scan is in progress, false otherwise

3.93.3.3 GetBroadcastAddress()

```
uint TableScanService.GetBroadcastAddress (
            IPAddress HostAddress,
            IPAddress SubnetMask )  [inline]
```

Gets the broadcast address of the network the host is on as an unsigned integer.

Parameters

HostAddress	The host IP address
SubnetMask	The subnet mask of the network used by the host

Returns

The broadcast address of the host IP address as an unsigned integer

3.93.3.4 GetNetworkAddress()

```
uint TableScanService.GetNetworkAddress (
            IPAddress HostAddress,
            IPAddress SubnetMask )  [inline]
```

Gets the network address of the network the host is on as an unsigned integer.

Parameters

HostAddress	The host IP address
SubnetMask	The subnet mask of the network used by the host

Returns

The network address of the host IP adddress as an unsigned integer

3.93.3.5 ScanInterface()

```
async Task<ConcurrentQueue<TableContext>?> TableScanService.ScanInterface (
             NetworkInterface networkInterface )  [inline]
```

Scans the provided network interface for cutting tables.

Parameters

networkInterface	The network interface to scan for cutting tables

Returns

A task containing a nullable ConcurrentQueue, which contains the discovered tables in a TableContext object

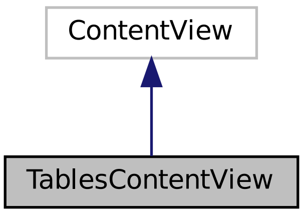
The documentation for this class was generated from the following file:

- Services/TableScanService.cs

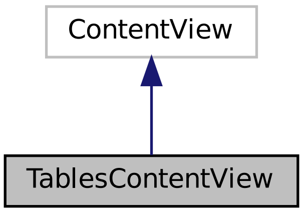
3.94 TablesContentView Class Reference

Content view for displaying tables.

Inheritance diagram for TablesContentView:



Collaboration diagram for TablesContentView:



Public Member Functions

- TablesContentView (MainViewModel mainViewModel)

  Initializes a new instance of the TablesContentView class.

3.94.1 Detailed Description

Content view for displaying tables.

Author

Ole William Skistad Huslende

Elvin Andreas Pedersen

3.94.2 Constructor & Destructor Documentation

3.94.2.1 TablesContentView()

```
TablesContentView.TablesContentView (
            MainViewModel mainViewModel )  [inline]
```

Initializes a new instance of the TablesContentView class.

Parameters

mainViewModel	Dependency injection of the main view model.

The documentation for this class was generated from the following file:

- Views/TablesContentView.xaml.cs

3.95 TableSerializationStructure Class Reference

Used to make a temporary list of tables used only for saving and loading tables on disk. Excludes information that should be stored securely (credentials) and information that can be retrieved again upon re-establishing the connection to the table

Properties

- string **Name** = string.Empty `[get, set]`
- string **Ip** = string.Empty `[get, set]`

3.95.1 Detailed Description

Used to make a temporary list of tables used only for saving and loading tables on disk. Excludes information that should be stored securely (credentials) and information that can be retrieved again upon re-establishing the connection to the table

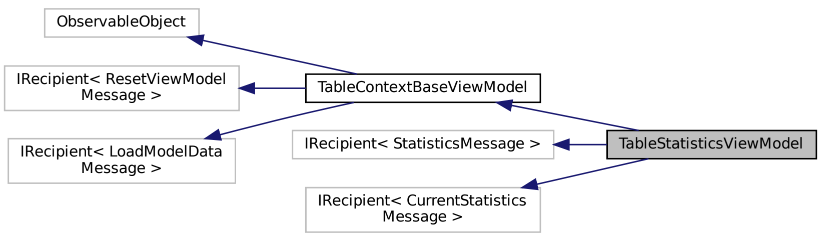
The documentation for this class was generated from the following file:

- Services/TableContextService.cs

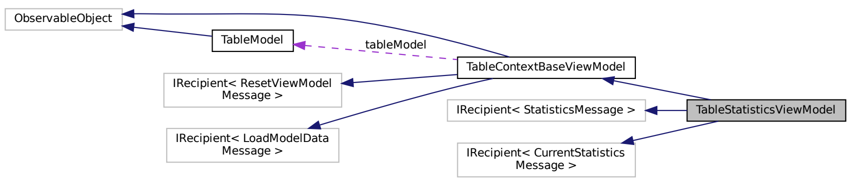
3.96 TableStatisticsViewModel Class Reference

The view model for the table statistics.

Inheritance diagram for TableStatisticsViewModel:


ObservableObject
IRecipient< ResetViewModel Message >
TableContextBaseViewModel
IRecipient< LoadModelData Message >
IRecipient< StatisticsMessage >
TableStatisticsViewModel
IRecipient< CurrentStatistics Message >

Collaboration diagram for TableStatisticsViewModel:


ObservableObject
TableModel
tableModel
TableContextBaseViewModel
IRecipient< ResetViewModel Message >
IRecipient< StatisticsMessage >
TableStatisticsViewModel
IRecipient< LoadModelData Message >
IRecipient< CurrentStatistics Message >

Public Member Functions

- TableStatisticsViewModel (IServiceProvider serviceProvider, IMessenger tableContextMessenger, IMessenger metaMessenger, TableModel tableModel)

    The constructor for the table statistics view model.
- void Receive (StatisticsMessage message)

    Receives a message containing statistics data and processes it.
- void Receive (CurrentStatisticsMessage message)

    Receives a message from the meta context containing the time span for the statistics.
- void GetDayStatistics ()

    Gets the statistics for the table for the last day.
- void GetWeekStatistics ()

    Gets the statistics for the table for the last week.
- void GetMonthStatistics ()

    Gets the statistics for the table for the last month.
- async void RequestProductionStatistics (DateTime before, DateTime after)

    Requests the production statistics for the table.
- string GenerateStatisticsCsv ()

    Generates a CSV file with the statistics for the table.

Static Public Member Functions

- static double GetUtilizationScore (TimeSpan totalBusyTime, TimeSpan totalTime)

    Calculates the utilization score for the table.

Protected Member Functions

- override void OnResetViewModel ()

    Resets the view model to its initial state.

Additional Inherited Members

3.96.1 Detailed Description

The view model for the table statistics.

Gathers and processes statistics data from the Kongsberg table. The data collected includes production time, utilization score, and job statistics.

Author

Ole William Skistad Huslende

3.96.2 Constructor & Destructor Documentation

3.96.2.1 TableStatisticsViewModel()

```
TableStatisticsViewModel.TableStatisticsViewModel (
             IServiceProvider serviceProvider,
             IMessenger tableContextMessenger,
             IMessenger metaMessenger,
             TableModel tableModel )  [inline]
```

The constructor for the table statistics view model.

This constructor initializes the view model with the service provider, messengers, and table model. Sets up the column definitions for the job, material, and customer statistics. Registers the view model to receive messages from the table context and meta context.

Parameters

serviceProvider	The service provider used to resolve dependencies.
tableContextMessenger	The messenger used to send messages between view models inside the table context.
metaMessenger	The messenger used to send messages between view models outside the table context.
tableModel	The table model used to get the table properties.

3.96.3 Member Function Documentation

3.96.3.1 GenerateStatisticsCsv()

```
string TableStatisticsViewModel.GenerateStatisticsCsv ( )  [inline]
```

Generates a CSV file with the statistics for the table.

Returns

Returns a string with the CSV file content.

3.96.3.2 GetDayStatistics()

```
void TableStatisticsViewModel.GetDayStatistics ( )  [inline]
```

Gets the statistics for the table for the last day.

3.96.3.3 GetMonthStatistics()

```
void TableStatisticsViewModel.GetMonthStatistics ( )  [inline]
```

Gets the statistics for the table for the last month.

3.96.3.4 GetUtilizationScore()

```
static double TableStatisticsViewModel.GetUtilizationScore (
             TimeSpan totalBusyTime,
             TimeSpan totalTime )  [inline], [static]
```

Calculates the utilization score for the table.

Parameters

totalBusyTime	The total time the table was busy during the period requested.
totalTime	The total time for the period requested.

Returns

Returns the utilization score for the table as a percentage.

3.96.3.5 GetWeekStatistics()

```
void TableStatisticsViewModel.GetWeekStatistics ( )  [inline]
```

Gets the statistics for the table for the last week.

3.96.3.6 OnResetViewModel()

```
override void TableStatisticsViewModel.OnResetViewModel ( )  [inline], [protected], [virtual]
```

Resets the view model to its initial state.

Reimplemented from TableContextBaseViewModel.

3.96.3.7 Receive() [1/2]

```
void TableStatisticsViewModel.Receive (
             CurrentStatisticsMessage message )  [inline]
```

Receives a message from the meta context containing the time span for the statistics.

Parameters

message	The message containing the time span for the statistics.

3.96.3.8 Receive() [2/2]

```
void TableStatisticsViewModel.Receive (
            StatisticsMessage message )  [inline]
```

Receives a message containing statistics data and processes it.

Gets messages from the table model containing statistics data. Resets the viewmodel to its initial state and processes the statistics data. If the request comes from the meta context, it sends a message to the meta context with the statistics data.

Parameters

message	

3.96.3.9 RequestProductionStatistics()

```
async void TableStatisticsViewModel.RequestProductionStatistics (
            DateTime before,
            DateTime after )  [inline]
```

Requests the production statistics for the table.

Parameters

before	The statistics requested before this time.
after	The statistics requested after this time.

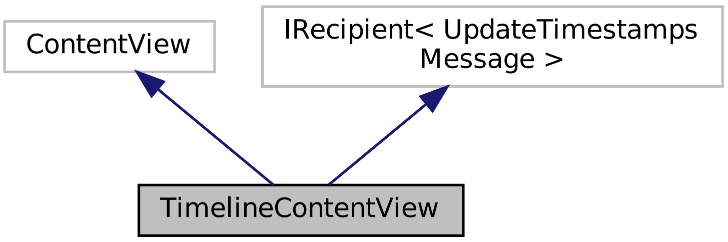
The documentation for this class was generated from the following file:

- ViewModels/TableContextViewModels/TableStatisticsViewModel.cs

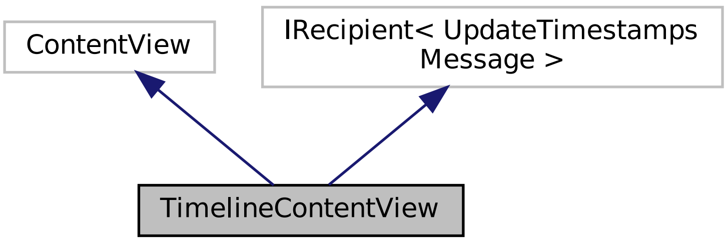
3.97 TimelineContentView Class Reference

Timeline for the active job list of a table. Shows job name, preview, duration and when they will finish

Inheritance diagram for TimelineContentView:



Collaboration diagram for TimelineContentView:



Public Member Functions

- TimelineContentView ()

    Constructor. Initializes the view, sets the binding context of the timestamp list.
- void Receive (UpdateTimestampsMessage message)

    Called when metamessenger sends an UpdateTimestampsMessage

Protected Member Functions

- override void OnBindingContextChanged ()

    Called when the binding context changes. Used to set bindingcontext to the viewmodel since contentviews cannot have parameters in their constructors

Properties

- ObservableCollection< Timestamp > Timestamps = [ ] `[get]`

    List of timestamps of when each job ends
- ObservableCollection< JobTaskViewModel >? JobTasks = null `[get, set]`

    List of JobTaskViewModel, for getting job times

3.97.1 Detailed Description

Timeline for the active job list of a table. Shows job name, preview, duration and when they will finish

Author

Tormod Smidesang

3.97.2 Constructor & Destructor Documentation

3.97.2.1 TimelineContentView()

```
TimelineContentView.TimelineContentView ( )  [inline]
```

Constructor. Initializes the view, sets the binding context of the timestamp list.

3.97.3 Member Function Documentation

3.97.3.1 OnBindingContextChanged()

```
override void TimelineContentView.OnBindingContextChanged ( )  [inline], [protected]
```

Called when the binding context changes. Used to set bindingcontext to the viewmodel since contentviews cannot have parameters in their constructors

3.97.3.2 Receive()

```
void TimelineContentView.Receive (
             UpdateTimestampsMessage message )  [inline]
```

Called when metamessenger sends an UpdateTimestampsMessage

Parameters

message	The message, required by the messenger system. Unused

3.97.4 Property Documentation

3.97.4.1 JobTasks

```
ObservableCollection<JobTaskViewModel>? TimelineContentView.JobTasks = null  [get], [set]
```

List of JobTaskViewModel, for getting job times

3.97.4.2 Timestamps

```
ObservableCollection<Timestamp> TimelineContentView.Timestamps = [ ]  [get]
```

List of timestamps of when each job ends

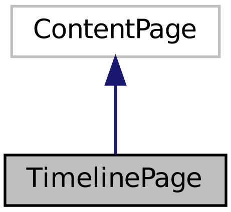
The documentation for this class was generated from the following file:

- Views/TimelineContentView.xaml.cs

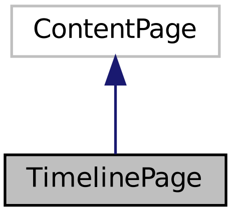
3.98 TimelinePage Class Reference

Timelinepage shows the timelines for the job tasks of the tables.

Inheritance diagram for TimelinePage:



Collaboration diagram for TimelinePage:



Public Member Functions

- TimelinePage (TimelineViewModel timelineViewModel)

  Initializes a new instance of the TimelinePage class.

Protected Member Functions

- override void OnAppearing ()

  Called when the page is appearing. This method is used to start the timestamp updater.
- override void OnDisappearing ()

  Called when the page is disappearing. This method is used to stop the timestamp updater.

3.98.1 Detailed Description

Timelinepage shows the timelines for the job tasks of the tables.

Author

Tormod Smidesang

3.98.2 Constructor & Destructor Documentation

3.98.2.1 TimelinePage()

```
TimelinePage.TimelinePage (
            TimelineViewModel timelineViewModel )  [inline]
```

Initializes a new instance of the TimelinePage class.

Parameters

timelineViewModel	Dependency injection of the view model for the timeline page.

3.98.3 Member Function Documentation

3.98.3.1 OnAppearing()

```
override void TimelinePage.OnAppearing ( )  [inline], [protected]
```

Called when the page is appearing. This method is used to start the timestamp updater.

3.98.3.2 OnDisappearing()

```
override void TimelinePage.OnDisappearing ( )  [inline], [protected]
```

Called when the page is disappearing. This method is used to stop the timestamp updater.

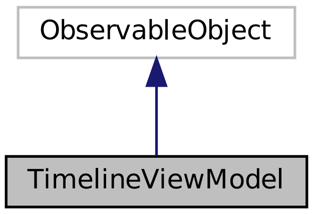
The documentation for this class was generated from the following file:

- Views/TimelinePage.xaml.cs

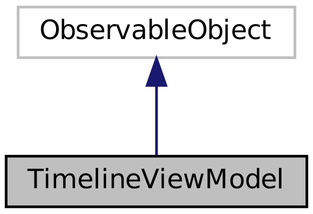
3.99 TimelineViewModel Class Reference

Viewmodel for the timeline page

Inheritance diagram for TimelineViewModel:



Collaboration diagram for TimelineViewModel:



Public Member Functions

- TimelineViewModel (TableContextService tableContextService, IMessenger metaMessenger)

  Constructor
- void OnPageAppearing ()

  Called by the code-behind of the page when the page is opened. Stops the timestamp updater
- void OnPageDisappearing ()

  Called by the code-behind of the page when the page is closed. Stops the timestamp updater

Properties

- IReadOnlyList< TableContext > Tables `[get]`

  List of tables from the tablecontextservice
- IMessenger MetaMessenger `[get]`

  The metamessenger, used to send messages to update the timestamps

3.99.1 Detailed Description

Viewmodel for the timeline page

Author

Tormod Smidesang

3.99.2 Constructor & Destructor Documentation

3.99.2.1 TimelineViewModel()

```
TimelineViewModel.TimelineViewModel (
            TableContextService tableContextService,
            IMessenger metaMessenger )  [inline]
```

Constructor

Parameters

tableContextService	The tablecontextservice, used to get a list of tables
metaMessenger	The metamessenger

3.99.3 Member Function Documentation

3.99.3.1 OnPageAppearing()

```
void TimelineViewModel.OnPageAppearing ( )  [inline]
```

Called by the code-behind of the page when the page is opened. Stops the timestamp updater

3.99.3.2 OnPageDisappearing()

```
void TimelineViewModel.OnPageDisappearing ( )  [inline]
```

Called by the code-behind of the page when the page is closed. Stops the timestamp updater

3.99.4 Property Documentation

3.99.4.1 MetaMessenger

```
IMessenger TimelineViewModel.MetaMessenger  [get]
```

The metamessenger, used to send messages to update the timestamps

3.99.4.2 Tables

```
IReadOnlyList<TableContext> TimelineViewModel.Tables  [get]
```

List of tables from the tablecontextservice

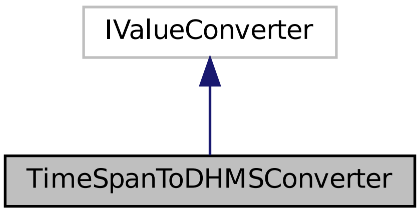
The documentation for this class was generated from the following file:

- ViewModels/TimelineViewModel.cs

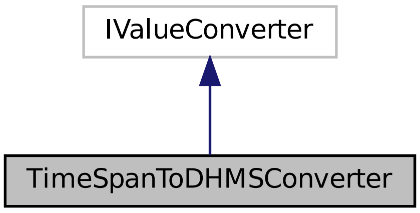
3.100 TimeSpanToDHMSConverter Class Reference

Converts a TimeSpan into a string that looks something like this: 3d6h34m3s. Changing the app language might change the symbols inbetween the numbers.

Inheritance diagram for TimeSpanToDHMSConverter:



Collaboration diagram for TimeSpanToDHMSConverter:



Public Member Functions

- object? Convert (object? value, Type targetType, object? parameter, CultureInfo culture)

  Converts a TimeSpan into a string that looks something like this: 3d6h34m3s.
- object? ConvertBack (object? value, Type targetType, object? parameter, CultureInfo culture)

  Not implemented

3.100.1 Detailed Description

Converts a TimeSpan into a string that looks something like this: 3d6h34m3s. Changing the app language might change the symbols inbetween the numbers.

Author

Tormod Smidesang

3.100.2 Member Function Documentation

3.100.2.1 Convert()

```
object?  TimeSpanToDHMSConverter.Convert (
             object?  value,
             Type targetType,
             object?  parameter,
             CultureInfo culture )  [inline]
```

Converts a TimeSpan into a string that looks something like this: 3d6h34m3s.

Parameters

value	A TimeSpan object to convert
targetType	unused
parameter	unused
culture	unused

Returns

A string representing the same timespan

3.100.2.2 ConvertBack()

```
object?  TimeSpanToDHMSConverter.ConvertBack (
             object?  value,
             Type targetType,
             object?  parameter,
             CultureInfo culture )  [inline]
```

Not implemented

The documentation for this class was generated from the following file:

- Utilities/Converters/TimeSpanToDHMSConverter.cs

3.101 Timestamp Class Reference

Timestamp class containing string representations of the date (only set if the current timestamp represents a new day) and the time of day.

Properties

- string **NewDate** = string.Empty `[get, set]`
- string **TimeOfDay** = string.Empty `[get, set]`

3.101.1 Detailed Description

Timestamp class containing string representations of the date (only set if the current timestamp represents a new day) and the time of day.

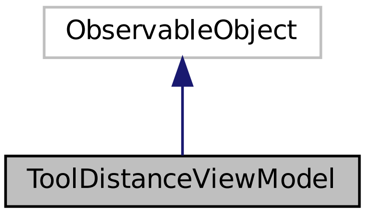
The documentation for this class was generated from the following file:

- Views/TimelineContentView.xaml.cs

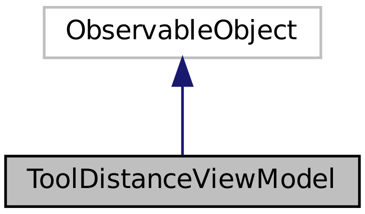
3.102 ToolDistanceViewModel Class Reference

ViewModel for displaying tool distance information.

Inheritance diagram for ToolDistanceViewModel:



Collaboration diagram for ToolDistanceViewModel:



3.102.1 Detailed Description

ViewModel for displaying tool distance information.

Author

Ole William Skistad Huslende

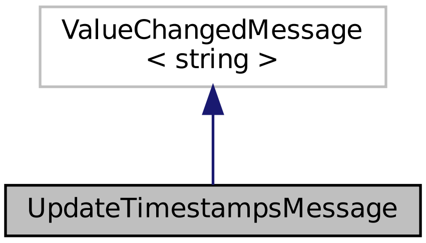
The documentation for this class was generated from the following file:

- ViewModels/TableContextViewModels/SubViewModels/ToolDistanceViewModel.cs

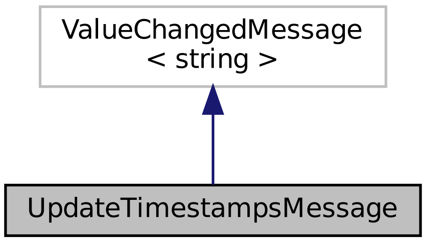
3.103 UpdateTimestampsMessage Class Reference

Message sent to the timeline page to update the timestamps.

Inheritance diagram for UpdateTimestampsMessage:



Collaboration diagram for UpdateTimestampsMessage:



Public Member Functions

- **UpdateTimestampsMessage** (string value)

3.103.1 Detailed Description

Message sent to the timeline page to update the timestamps.

Author

Tormod Smidesang

The documentation for this class was generated from the following file:

- ViewModels/ViewModelMessages.cs

3.104 UserPreferences Class Reference

Service responsible for storing user preferences. TODO: remove this class and use the Preferences API built into .NET instead

Public Member Functions

- UserPreferences ()

  Constructor. Sets CultureName to the current culture.
- void Save ()

  Saves the user preferences to disk.
- void ApplyCulture ()

  Applies the culture set by the user. Must be called before a page is initialized or changes will not be reflected.

Static Public Member Functions

- static UserPreferences Load ()

  Loads the preferences from disk.

Properties

- string CultureName `[get, set]`

  The culture name specified by the user. Example value: "en-US".

3.104.1 Detailed Description

Service responsible for storing user preferences. TODO: remove this class and use the Preferences API built into .NET instead

Author

Tormod Smidesang

3.104.2 Constructor & Destructor Documentation

3.104.2.1 UserPreferences()

```
UserPreferences.UserPreferences ( )  [inline]
```

Constructor. Sets CultureName to the current culture.

3.104.3 Member Function Documentation

3.104.3.1 ApplyCulture()

```
void UserPreferences.ApplyCulture ( )  [inline]
```

Applies the culture set by the user. Must be called before a page is initialized or changes will not be reflected.

3.104.3.2 Load()

```
static UserPreferences UserPreferences.Load ( )  [inline], [static]
```

Loads the preferences from disk.

Returns

A UserPreferences object containing the user preferences stored on disk

Exceptions

FileNotFoundException	Thrown if filePath does not exist

3.104.3.3 Save()

```
void UserPreferences.Save ( )  [inline]
```

Saves the user preferences to disk.

3.104.4 Property Documentation

3.104.4.1 CultureName

```
string UserPreferences.CultureName  [get], [set]
```

The culture name specified by the user. Example value: "en-US".

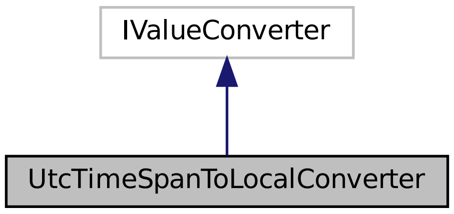
The documentation for this class was generated from the following file:

- Services/UserPreferences.cs

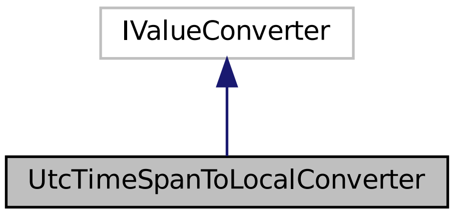
3.105 UtcTimeSpanToLocalConverter Class Reference

Converts UTC TimeSpan to time of day in local time.

Inheritance diagram for UtcTimeSpanToLocalConverter:



Collaboration diagram for UtcTimeSpanToLocalConverter:



Public Member Functions

- object? Convert (object? value, Type targetType, object? parameter, CultureInfo culture)

  Converts UTC TimeSpan to time of day in local time.
- object? ConvertBack (object? value, Type targetType, object? parameter, CultureInfo culture)

  Converts local time TimeSpan to time of day in local time.

3.105.1 Detailed Description

Converts UTC TimeSpan to time of day in local time.

Author

Ole William Skistad Huslende

3.105.2 Member Function Documentation

3.105.2.1 Convert()

```
object?  UtcTimeSpanToLocalConverter.Convert (
             object?  value,
             Type targetType,
             object?  parameter,
             CultureInfo culture )  [inline]
```

Converts UTC TimeSpan to time of day in local time.

Parameters

value	UTC TimeSpan
targetType	unused
parameter	unused
culture	unused

Returns

A string representation of the time of day in the TimeSpan in local time

3.105.2.2 ConvertBack()

```
object?  UtcTimeSpanToLocalConverter.ConvertBack (
             object?  value,
             Type targetType,
             object?  parameter,
             CultureInfo culture )  [inline]
```

Converts local time TimeSpan to time of day in local time.

Parameters

value	Local time TimeSpan
targetType	unused
parameter	unused
culture	unused

Returns

UTC TimeSpan representation of the given local time TimeSpan

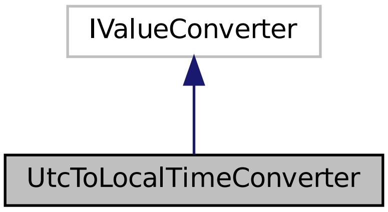
The documentation for this class was generated from the following file:

- Utilities/Converters/UtcTimeSpanToLocalConverter.cs

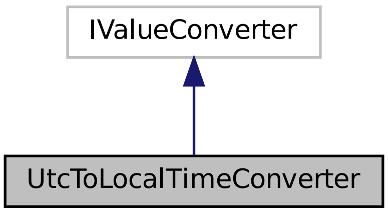
3.106 UtcToLocalTimeConverter Class Reference

Converts UTC time to local time.

Inheritance diagram for UtcToLocalTimeConverter:



Collaboration diagram for UtcToLocalTimeConverter:



Public Member Functions

- object Convert (object value, Type targetType, object parameter, CultureInfo culture)

  Converts UTC time to local time.
- object ConvertBack (object value, Type targetType, object parameter, CultureInfo culture)

  Not implemented

3.106.1 Detailed Description

Converts UTC time to local time.

Author

Elvin Andreas Pedersen

3.106.2 Member Function Documentation

3.106.2.1 Convert()

```
object UtcToLocalTimeConverter.Convert (
             object value,
             Type targetType,
             object parameter,
             CultureInfo culture )  [inline]
```

Converts UTC time to local time.

Parameters

value	A DateTime object in UTC time
targetType	unused
parameter	unused
culture	unused

Returns

A string representation of the same time in local time

3.106.2.2 ConvertBack()

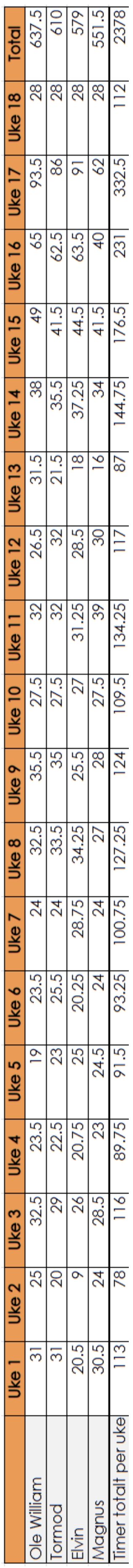
```
object UtcToLocalTimeConverter.ConvertBack (
             object value,
             Type targetType,
             object parameter,
             CultureInfo culture )  [inline]
```

Not implemented

The documentation for this class was generated from the following file:

- Utilities/Converters/TimeZoneConverter.cs

L Timesheet



	Uke 1	Uke 2	Uke 3	Uke 4	Uke 5	Uke 6	Uke 7	Uke 8	Uke 9	Uke 10	Uke 11	Uke 12	Uke 13	Uke 14	Uke 15	Uke 16	Uke 17	Uke 18	Total
Ole William	31	25	32.5	23.5	19	23.5	24	32.5	35.5	27.5	32	26.5	31.5	38	49	65	93.5	28	637.5
Tormod	31	20	29	22.5	23	25.5	24	33.5	35	27.5	32	32	21.5	35.5	41.5	62.5	86	28	610
Elvin	20.5	9	26	20.75	25	20.25	28.75	34.25	25.5	27	31.25	28.5	18	37.25	44.5	63.5	91	28	579
Magnus	30.5	24	28.5	23	24.5	24	24	27	28	27.5	39	30	16	34	41.5	40	62	28	551.5
Timer totalt per uke	113	78	116	89.75	91.5	93.25	100.75	127.25	124	109.5	134.25	117	87	144.75	176.5	231	332.5	112	2378

Figure 39: Timesheet