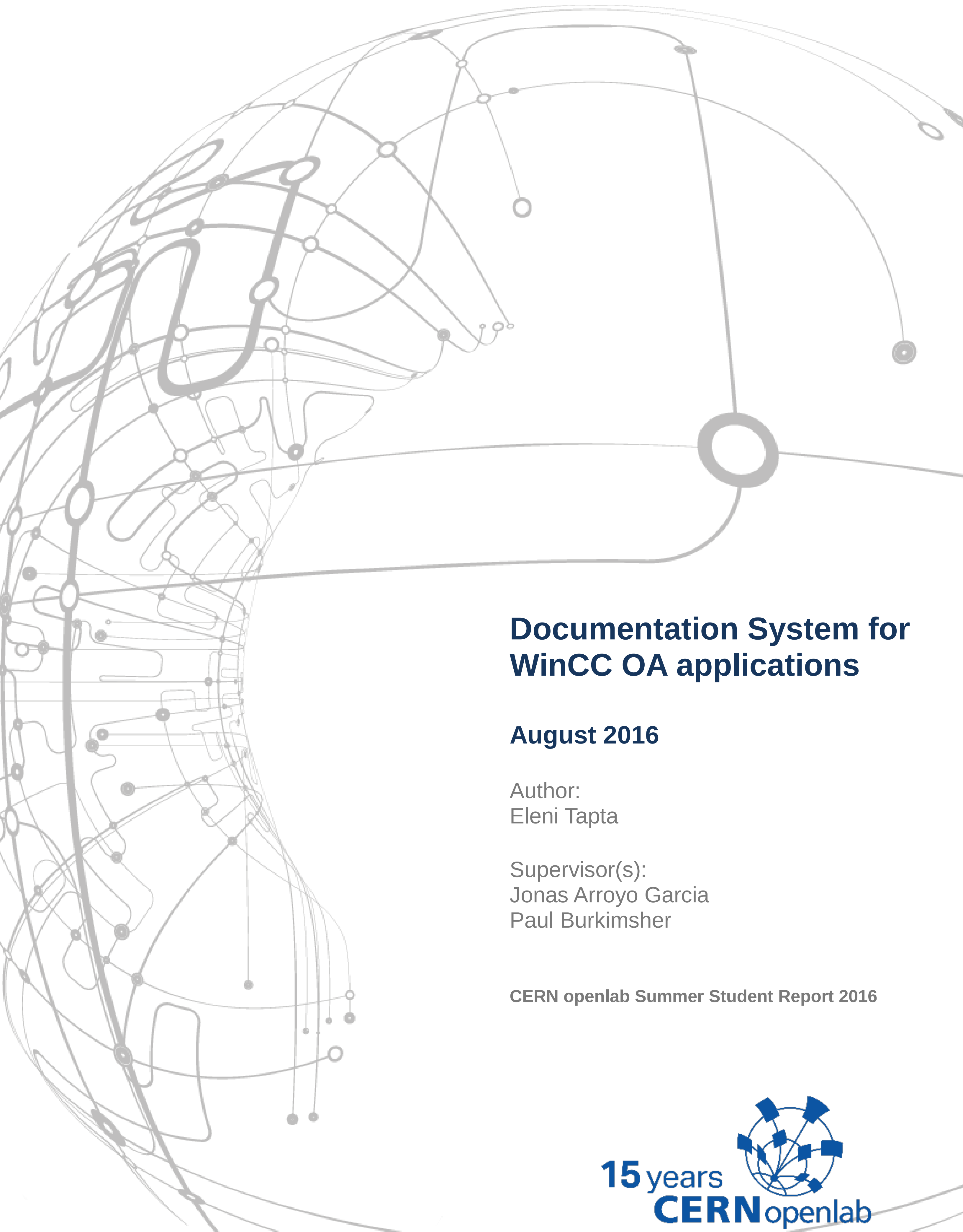


Documentation System for WinCC OA applications

August 2016

Author:
Eleni Tapta

Supervisor(s):
Jonas Arroyo Garcia
Paul Burkimsher

CERN openlab Summer Student Report 2016

Project Specification

The project focuses on the evaluation of different technologies for the documentation of the JCOP and UNICOS frameworks software components (based on WinCC Open Architecture)

In this scope propose a help system for; (1) WinCC OA developers to easily document and consult generated documentation; (2) WinCC OA users in the applications to access the application help

Abstract

Most of the control system applications that are used at CERN are developed with the use of different pieces of software. One of them is WinCC OA, a commercial SCADA tool developed by Siemens. The others are frameworks, like JCOP and UNICOS that were written at CERN and are built and run under the context of WinCC OA. The other one is the JCOP framework, a tool that is written at CERN and is built and runs under the context of WinCC OA. However, while these tools are co-dependent, they have completely separate sets of documentation. WinCC OA, in particular, uses a powerful help system that is built with tools that are provided by the Qt and Qt help framework. On the other hand, JCOP uses Doxygen to generate its documentation from annotated code sources. This documentation is later displayed in HTML pages that are installed locally in the computer and provide very little functionality. The main challenges of this project are as follows (1) to find a way to properly integrate the existent documentation into the help browser that is used by WinCC OA; (2) while also finding solutions that would give the help system more functionalities than were available before.

Keywords: WinCC OA, SCADA, JCOP, UNICOS, Doxygen, Qt, Qt help

Table of Contents

1 Introduction

1.1 WinCC OA

WinCC-OA is a SCADA system that is used for the visualization and the operation of processes and production flows in many different fields of industry or research. It was originally developed at ETM under the name PVSS, but currently both the software and the company are owned by SIEMENS. At CERN this software is used for the development of most of the control systems that run around the site. This however, does not mean that WinCC-OA is a standalone control system itself. It actually works as a tool that lets the developers implement their own control systems on top of it. A few examples of such systems and use cases are the control of the Quench Protection System for the super-conducting magnets and the cryogenics for the LHC accelerator, CERN electrical network and many others.

WinCC OA is a modularly built system. This means that the required functionalities are handled by specific units that were created for different tasks. In WinCC OA these units are called managers which run in different layers of the system.

One of the managers that was mostly important for this project was the Graphic Editor, also known as GEDI. In general, this module provides tools for creating and configuring graphic objects. These can be graphical representations of complete systems or just simple operating screens. One of the tools in GEDI is the Script Editor which is used for creating and editing scripts and libraries with the use of CTRL, a language similar to C. As far as the services of the WinCC OA help system are concerned, one of its capabilities is to access the documentation of a CTRL function while editing the code, just by right-clicking on its name. However, describing the rest of the features supported by GEDI or every other manager in detail goes beyond the scope of this report.

1.2 JCOP Framework

JCOP (Joint Controls Project) is a framework which is built at CERN under the context of WinCC OA and is used by the developers to develop their part of the control system application. In general, JCOP aims towards the reduction of the overall manpower cost that is required to produce and run the various control systems for the experiments. For this reason, it provides many tools and libraries that hide a lot of the complexity of WinCC-OA, as well as guidelines and conventions for the easier integration of software from many labs. This ultimately leads to homogenous control systems that are easily maintained and transferred from one experiment to another, since they were implemented in the same ways.

1.2.1 JCOP Component Installation Tool

Similarly to the WinCC OA, the JCOP framework should not be perceived as a complete individual system, but rather as a set of software components that can communicate and work together. Even during the implementation of the applications, not all possible components are installed, but merely those needed by an individual project.

In fact, the installation of software component in WinCC OA-based applications is managed by the Installation Tool Component of the JCOP framework. When installing a new component, this tool makes use of an XML file which contains information about that particular component, such as name, version, paths of the files used by the component etc. A few of the tags that are supported in that file are described below.

- Help: The help tag specifies the name of the help file to be opened from the main panel of the Installation Tool. For example:

```
<help>./fwAccessControl/fwAccessControl.htm</help>
```

- postInstall: The filename of a script that should run right after the installation of the component. For example:

```
<postInstall>./config/fwDIM.postInstall</postInstall>
```

- postDelete: Similar to the one describe above, this tag will specify a script to be executed after the deletion of a component. For example:

```
<postDelete>./myPostDelete.ctl</postDelete>
```

1.3 Doxygen

Doxygen is a tool for generating documentation from annotated code sources. It mainly supports C-like languages, but it can also work with other programming languages, such as Java, Python etc. Its primary purpose is the automatic generation of complete software manuals in many different formats, such as online HTML documentation, hyperlinked PDF, LATEX, RTF (MS-Word) and others. Doxygen can also be used to extract the code structure and then visualize the relations between the various elements with dependency graphs, inheritance diagrams and collaboration diagrams. This additional feature can be proven useful, especially in large source distributions or in undocumented source files.

Practically, in order to create a manual for a project with Doxygen, there are certain requirements that need to be taken into account. Firstly, the user should make sure that the source code is documented with special comments that will be recognized by Doxygen as documentation blocks. Doxygen, will then need a configuration file that will determine the settings of the manual. The format of this file is very similar to that of a simple Makefile and it consists of a number of assignments in the following form: TAGNAME=VALUE. To simplify the creation of such a configuration file a template can be created with the command:

```
doxygen -g <config_filename>
```

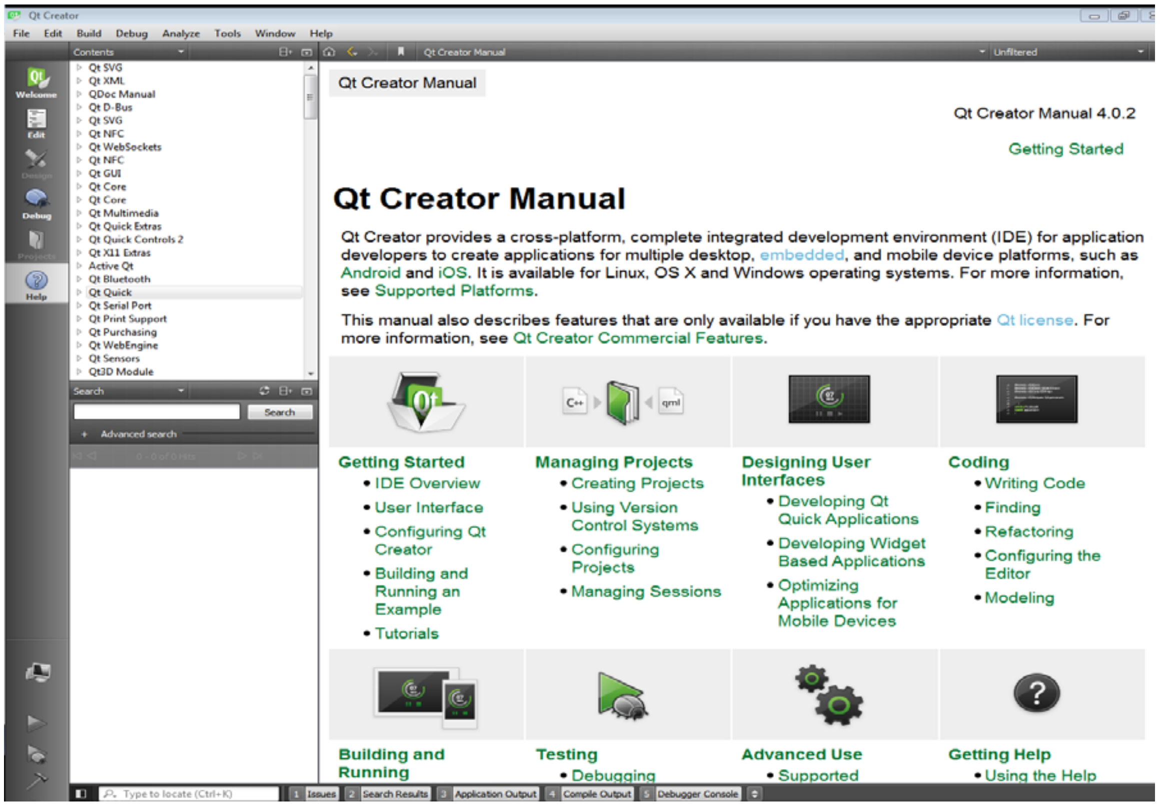
After the generation and the adjustment of the configuration file, the executable doxygen can be called from the command line, in order to parse the sources and generate the documentation. Alternatively, the executable Doxywizard can be used, which is a graphical front-end that can be used both for editing the configuration file and for running doxygen in a graphical environment.

1.4 Qt Framework

1.4.1 Qt Assistant

Qt is a cross-platform application framework, which is widely used for the development of application software that can be run on various software and hardware platforms.

It comes with its own set of tools for easier cross-platform development, such as its own IDE – Qt Creator or the interface builder, Qt Designer. One of the tools that are included in a Qt installation is also the Qt Assistant, which is a help browser that is embedded in Qt Creator and can be used for viewing online documentation in Qt help file format.



Qt Creator Manual

Qt Creator Manual 4.0.2

Getting Started

Qt Creator Manual

Qt Creator provides a cross-platform, complete integrated development environment (IDE) for application developers to create applications for multiple desktop, embedded, and mobile device platforms, such as Android and iOS. It is available for Linux, OS X and Windows operating systems. For more information, see Supported Platforms.

This manual also describes features that are only available if you have the appropriate Qt license. For more information, see Qt Creator Commercial Features.

Getting Started
- IDE Overview
- User Interface
- Configuring Qt Creator
- Building and Running an Example
- Tutorials

Managing Projects
- Creating Projects
- Using Version Control Systems
- Configuring Projects
- Managing Sessions

Designing User Interfaces
- Developing Qt Quick Applications
- Developing Widget Based Applications
- Optimizing Applications for Mobile Devices

Coding
- Writing Code
- Finding
- Refactoring
- Configuring the Editor
- Modeling

Building and Running

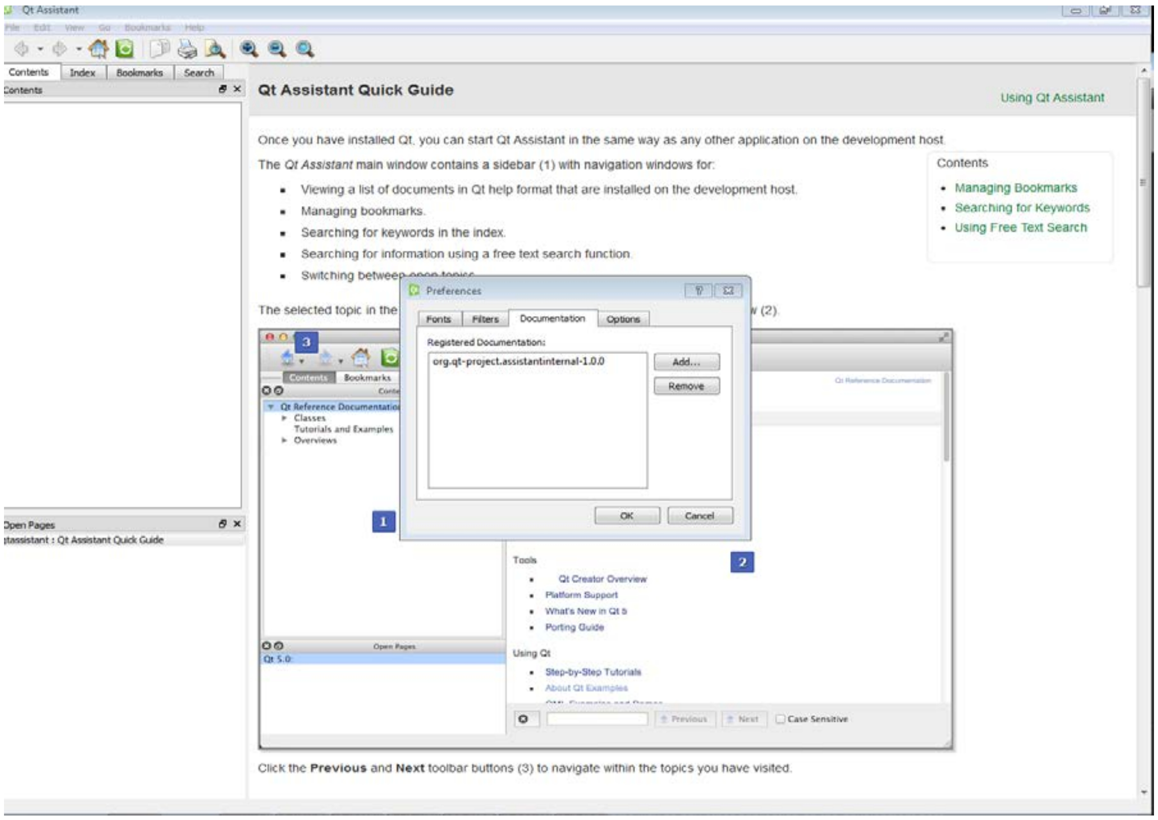
Testing
- Debugging

Advanced Use
- Supported

Getting Help
- Using the Help

Figure 1. Main panel of the Qt Assistant in the Qt Creator.

While Qt installations always include a standard set of Qt documentation files, it is always possible to add custom documentation to the set of documents that are displayed in the Qt Assistant's main view. As it is displayed in the screenshot below, in order to add an external help file manually, we have to open the Preferences window from the Edit menu (see *Figure 2*)



Qt Assistant Quick Guide

Once you have installed Qt, you can start Qt Assistant in the same way as any other application on the development host.

The *Qt Assistant* main window contains a sidebar (1) with navigation windows for:

- Viewing a list of documents in Qt help format that are installed on the development host.
- Managing bookmarks.
- Searching for keywords in the index.
- Searching for information using a free text search function.
- Switching between open topics.

The selected topic in the … w (2).

Click the **Previous** and **Next** toolbar buttons (3) to navigate within the topics you have visited.

Registered Documentation: org.qt-project.assistantinternal-1.0.0

Figure 2. Preferences and Documentation windows in the Qt Assistant.

Furthermore, the Assistant itself can be easily customized and used as a help viewer to any applications, even those not developed with the Qt Framework. In particular, it is important to be able to customize the appearance of the help window, such as the icon or the title. An additional requirement for such a help system, would be the ability to receive actions or commands from the application it provides help for. WinCC OA, in particular, can provide context-sensitive help. This means that it can call for a certain help file to be displayed depending on the user's requests. For instance, the Help menu in GEDI contains certain options regarding the help that should be displayed when starting the help browser, such as Gedi Module, CTRL module and others. If the user clicks on the Gedi Module, then the page that the Qt Assistant will display on start, will be this one, instead of the usual WinCC OA homepage.

1.4.2 Qt help system

In order to generate custom documentation in Qt help file format, the Qt Help framework must be used. This framework does not only include many tools for producing the help files, but it also provides classes for accessing help contents programmatically and integrating online help into Qt applications.

When generating documentation with Qt, the actual help data, such as the table of contents, index keywords or HTML documents are contained in Qt compressed help files. This means that there is no need to ship all single HTML files. Instead only the compressed help file has to be distributed. Each Qt compressed help file is associated with a different file called Qt help project.

The latter has an XML format and is the file which collects all the necessary data. Along with the actual help data, it contains some extra information like a unique namespace to identify the help file. So, such a help file represents usually a single manual or documentation set.

Since most products are more comprehensive and consist of a number of tools, like WinCC-OA or JCOP, one manual is rarely enough. Instead, more manuals which should be accessible at the same time, exist. Ideally, it should also be possible to reference certain points of interest of one manual to another. Therefore, the Qt help system operates on help collection files which can include any number of compressed help files that can be collectively shipped to the Qt Assistant.

On the downside, having collection files to merge many documentation sets may lead to some problems. For example, one index keyword may be defined in different documentations and lead to conflicts in the Qt Assistant. So, when only seeing it in the index and activating or searching it, the user cannot be sure that the expected documentation will be shown. Therefore, the Qt help system offers the possibility to filter the help contents after certain attributes. This requires however, that the attributes have been assigned to the help contents before the generation of the compressed help file.

So, in general, there are four files interacting with the help system: two used for generating Qt help and two meant for distribution. All of the information described above can be easily presented in the following tables.

Name file	***Qt help project***	***Qt help collection project***
Extension	.qhp	.qhcp
Brief Description	The input file for the help generator consisting of the table of contents, indices and references to the actual documentation files (*.html); it also defines a unique namespace for the documentation.	The input file for the help collection generator. It contains references to compressed help files which should be included in the collection; it also may contain other information for customizing Qt Assistant.

```
$ qhelpgenerator ~.qhp -o ~.qch
```

```
$ qcollectiongenerator ~.qhcp -o ~.qhc
```

Name file	**Qt Compressed help**	**Qt Help Collection**
Extension	.qch	.qhc
Brief Description	The output file of the help generator. This binary file contains all information specified in the help project file along with all compressed documentation files.	The output of the help collection generator. This is the file QHelpEngine operates on. It contains references to any number of compressed help files as well as additional information, such as custom filters.

Finally, it is important to note that both the compressed help (.qch) and collection (.qhc) files can also be handled as sqlite databases. This means that using sqlite3 and with the appropriate commands, the help data associated with the project files can be easily retrieved from the binary help files. This small piece of information can be proven very useful in cases were the developer needs to make modifications in certain compressed help or collection files but does not have access to the original projects.

2 Project Description

2.1 Motivation

As it has already been mentioned above, the developers use WinCC-OA and the GEDI user interface to develop their own applications. Siemens has recently changed the way they generate and display the WinCC-OA documentation and has migrated to Qt help and Qt Assistant. However, the documentation for all the software libraries of each JCOP component is still presented in individual static HTML pages which were generated with Doxygen and are accessible only via the local installation directory of each component. It is therefore evident that this situation is far from being optimal, not only because the JCOP help is not integrated with the one of Qt, but also because the HTML pages, being static, provide very little functionality. One of the most characteristic examples of the limited range of capabilities of the Doxygen files, is the fact that it is not possible to search or cross-reference topics between components. In the view of such limitations, the developers can be much more reluctant to actually use or write the documentation.

2.2 Objective

Nevertheless, the fact that the WinCC OA documentation is now generated and displayed with Qt tools, opens up many possibilities to set up an appropriate help system for the JCOP and UNICOS frameworks. While, certainly the main objective is to properly integrate the documentation into the help system of WinCC OA, there are also many other suggestions that would make the system even more convenient and appealing for the potential users.

Firstly, one of the key tasks would be to properly define index keywords on functions in order to improve the functionality of the search index tool provided by Qt Assistant, as well as the full text searching.

Furthermore, another helpful suggestion would be to integrate the JCOP documentation into the GEDI script editor. Actually, this functionality is already available for the WinCC-OA functions, where you can access the corresponding documentation for each function simply by right clicking on its name.

Finally, since the documentation of WinCC-OA and JCOP combined lead to an extensive manual, it would be useful to be able to define custom filters and attributes. The intention is to filter all of the documentation depending on the version of each component or on a particular project and its respective installed components. This practice would not only make the documentation better organized and easily accessible, but could also lead to getting faster and more reliable search results. For instance, in many cases the name of a certain element can be found in multiple pages. These pages could either be associated with the documentation of different versions of the same component or with the various help files that exist in the collection in general. Meanwhile, Qt Assistant orders the list of the search results according to the number of occurrences of the text in the documents, with those containing the highest number of occurrences appearing first. This means that while searching for a term, there is a big possibility of getting the wrong top results, which can make the search process unnecessarily complicated and lengthy, especially in big manuals, or when including the manual for different versions of the same component.

To sum up the main objectives of this project will be:

- Embed the JCOP documentation into the WinCC OA documentation and Qt Assistant
- Add proper indexes and keywords for the function
- Integrate the JCOP documentation into the GEDI script editor by WinCC OA
- Filter the documentation based on attributes related to the various projects or component versions

3 Methodology for the implementation

3.1 Fw Components

Firstly, it is important to download some of the JCOP components such as the fwTrending, fwAccessControl etc, so that we can use them as a base for Doxygen and test the outcome of the project in general. Normally, these components are not installed in the machine, but in the specific installation directories of the project that uses them. However, having multiple help files installed in the various project folders would be impractical. Therefore, it would be better to create a central working repository for the installation of the help folders. This means that all the documentation files were stored in a central repository in the C: drive and the help system would later pick up the files from there.

3.2 Doxygen

As it has been already mentioned above, in order to generate documentation from annotated source files with doxygen, a configuration file is necessary.

In our case, there was already a configuration file in the directories of the majority of the downloaded components. This was actually the old file based on which the current documentation files have been created.

However, there were still cases where the doxygen configuration file of the component did not exist or was not accessible. In that case it was necessary to use the command doxygen –g <config-file> to generate a template for the configuration file.

Once, the template was ready it needed to be actually configured according to the needs of the project. Since the components would follow the same generic style of documentation, most of the attributes of the configuration file were left with their default values. However, there were still attributes that needed to be explicitly defined for each component. Such cases were the name of the project, the output directory, the extension mapping and others which can be summed up in the following table.

TAGNAME	VALUE	USAGE
PROJECT_NAME	Identify the project. Usually it has the name of the corresponding component.	A single word (or a sequence of words surrounded by quotes) that should identify the project.
OUTPUT_DIRECTORY	Help/en_US.utf8/ Help/en_US.iso88591	The directory where the help files would be saved. Usually inside the component directory tree under /help/en_US.utf8
EXTENSION_MAPPING	.ctl=C	Override or extend Doxygen's built-in mapping and assign which language parser should be used for a given extension. In our case we are going to handle .ctl (extension) = C(parser)
FILE_PATTERNS	*.ctl	Specify a wildcard pattern to filter out the source-files in the directories. For the project we only needed to check .ctl files
GENERATE_QHP	YES	Used to create Qt help files, Value=yes
QCH_FILE	ComponentName.qch	Specify the name of the resulting .qch file. Usually the name of the component and the version
QHP_NAMESPACE	JCOP.componentname-version	Specify the namespace for the component documentation
QHP_VIRTUAL_FOLDER	JCOP	Specify the virtual folder for the component documentation

The table above can give the general idea about the way a doxygen configuration file would look, as well as the type of information that were needed for the generation of the JCOP html help pages. It's worth noting that there were many other attributes defined as well, regarding some of the extra functionalities (e.g. include source files, or order file members alphabetically), however only those that are particularly important should be included for the purpose of this report.

Once the configuration file is ready the executable doxygen has to be called. This is the program that parses the sources and generates the documentation according to the info specified by the configuration file.

```
$doxygen <configurationFile>
```

After the process is over, the html files along with the .qhp and the .qch files should be stored in the appropriate folders. The path of these directories should be registered in the XML file that is required by the fwInstallation Component, under the help tag. This final step is very important for the successful installation of the component in a future project.

3.3 Qt help

A Qt help project is the file that collects all the data necessary to generate Qt documentation. Along with the actual help data, like the table of contents or the html files, it also includes extra information like keywords and index ids.

Even though doxygen provides the necessary functions to generate qhp and qch files and can actually register most of the information correctly, it still provides little flexibility, especially when it comes to the definition of keywords or filters. Therefore, it is necessary to modify the doxygen-generated qhp file in a way that fits our requirements. The file format of that file is XML-based. Below, is a small example of the XML source that was written for the .qhp file for one of the components.

3.3.1 Namespace

```
<?xml version="1.0" encoding="UTF-8"?>
<QtHelpProject version="1.0">
      <namespace>jcop.fwTrending-7.0.0</namespace>
      <virtualFolder>JCOP</virtualFolder>
...
```

To enable the Qt Assistant to retrieve the proper documentation to a given link, every documentation set has to have a unique identifier. Furthermore, the unique identifier makes it possible for the help collection to keep track of a documentation set without relying on its file name. The Qt help system uses a name as defined in the mandatory namespace tags. In this example, the namespace is jcop.fwTrending-7.0.0. It is important to point out that the version of the component is also included, in order to be able to include the documentation for different versions, without them getting mixed.

3.3.2 Virtual Folders

Having a namespace for every documentation, as described above, naturally means that the documentation sets are quite separated. From the help engine's point of view this is beneficial, but from the documenter's view it is often desirable to cross reference certain topic from one manual to another without having to specify absolute links. To solve this problem, the help system introduced the concept of virtual folders. In our case the virtual folder is JCOP for all of the components, so that it enables the communication between their help files.

3.3.3 Custom filters

```
...
<customFilter name="Project A">
      <filterAttribute>projA</filterAttribute>
</customFilter>
<filterSection>
      <filterAttribute>projA</filterAttribute>
      <filterAttribute>fwTrending_7.0.0</filterAttribute>
...
```

A custom filter contains a list of filter attributes which will be used later to display only the documentation which has all those attributes. In this project the custom filters are related to the project name and the component version. So, when setting the current filter in the Assistant to "Project A" only the documentation which has "projA" set as filter attributes will be shown. Therefore, by adding projA as a filter attribute to the qhp files of the components that are used in Project A, we will be able to filter the documentation by the components that are installed in the project.

In general, it is possible to define any number of custom filters in a help project file. The important thing to know is, that the filter attributes do not have to be specified in the same project file; they can be defined in any other help file. The definition of a filter attributes takes place by specifying them in a filter section.

3.3.4 Filter Section

A filter section contains the actual documentation. One Qt help project file may contain more than one filter sections. Every filter section consists of four parts, the filter attributes section, the table of contents, the keywords and the files list. In theory all parts are optional but not specifying anything there will result in an empty documentation**.**

3.3.5 Filter Attributes

Every filter section should have filter attributes assigned to it, to enable documentation filtering. In this case, the filter attributes ‘projA’ and ‘fwTrending_7.0.0’ are assigned to the filter section, i.e. all contents specified in this section will only be shown if the current custom filter has ‘projA’ or ' fwTrending_7.0.0’ or both as filter attributes. This means that we will be able to filter the documentation that is displayed in the Qt Assistant according to the desired component version or a particular project.

3.3.6 Table of Contents

```
...
<toc>
   <section title="fwTrending-7.0.0. Component"
ref="fwTrending7.0.0/index.html">
      <section title="Files">
         <section title="File List" ref="fwTrending-
7.0.0/files.html">
     ...
</toc>
```

One section tag represents one item in the table of contents. The sections can be nested to any degree, but from a user's perspective it should not be more than four or five levels. A section is defined by its title and reference. The reference, like all file references in a Qt help project, are relative to the help project file itself.

3.3.7 Keywords

```
. . .
<keywords>
  <keyword name="fwTrending_getColRow"
id="fwTrending_getColRow" ref="fwTrending…ctl.html"/>
  . . .
</keywords>
<files>
   <file> fwTrending-7.0.0/tabs.css</file>
   . . .
</files>
</filtersection>
<QtHelpProject>
```

The keyword section lists all keywords of this filter section. A keyword consists basically of a name and a file reference. If the attribute 'name' is used then the keyword specified there will appear in the visible index, i.e. it will be accessible through the QHelpIndexModel.

If 'id' is used, the keyword does not appear in the index and is only accessible via the linksForIdentifier() function of the QHelpEngineCore. In the WinCC_OA help system, specifically, the keywords' ids are essential for the JCOP help integration in the GEDI script editor. ETM developers have already implemented this functionality for the WinCC OA documentation. It works by looking through every help file in the collection for the keyword with that specific name. This means that using the ids the GEDI script editor will be able to access the JCOP documentation.

In our case both attributes are specified, since both functionalities are needed.

3.3.8 Files

Finally, the actual documentation files have to be listed. Make sure that all files necessary to display the help are mentioned, i.e. stylesheets or similar files need to be there as well. The files, like all file references in a Qt help project, are relative to the help project file itself.

All listed files will be compressed and written to the Qt compressed help file. So, in the end, one single Qt help file contains all documentation files along with the contents and indices.

3.4 Collection File

3.4.1 WinCC_OA Collection

WinCC OA already has its own collection file that contains compressed help files. When using WinCC OA the user can access the documentation by clicking on an indicative UI button that opens the Qt Assistant. It is evident from the source files and the comments of WinCC_OA that the functions that are needed for the help display are implemented in such a way that the WinCC_OA collection file should always be available and under that specific name. This means that in order to install our own documentation in WinCC_OA we need a collection named WinCC_OA.

While one way around that would be to simply create a new collection project (qhcp) file and reregister all the WinCC OA qch files along with the JCOP ones with XML, the "safest" option is to register the JCOP documentation files in the existing WinCC_OA collection. This is necessary because WinCC OA collection would include more information in the Qhcp other than the basic file registration and extracting the correct information from a collection file as big as the one of WinCC OA can be very complicated, even with sqlite.

3.4.2 JCOP help registration

The simplest way to register manually a qch file into a collection is through the Qt Assistant, as it was described in previous chapters.

However, this is not useful in our case because we want to integrate the JCOP documentation not only in the Qt Assistant display, but also in the rest of the functions that were implemented by Siemens for the WinCC OA documentation (e.g. GEDI).

A workaround for this is to actually run the following commands for the registration of our documentation files.

```
$assistant -collectionFile WinCC_OA.qhc -register fwTrending.qch
```

After the registration process, we can always check if all the information was registered correctly by using sqlite. As it has already been mentioned, every help file can be handled as a sqlite database. Therefore we can actually open the collection database and see all the relevant tables for that database. After that we can select the data e.g. for the Namespace table (if we want to see which qch files are registered in the collection). We can always access the data of any of the tables, however it can be impractical depending on the amount of information stored and the size of the collection file.

```
$sqlite .open WinCC_OA.qhc
$sqlite .tables
$sqlite select * from NamespaceTable;
```

```
$FilterAttributeTable   FilterTable   NamespaceTable
$FilterNameTable        FolderTable   SettingsTable
```

```
$1|WinCC_OA|WinCC_OA.qch
$2|jcop.fwTrending-5.2.0|WinCC_OA_help/315/…fwTrending-
5.2.0.qch
```

4 Testing with Qt Assistant

Once we have completed the above process and have checked that the help files are registered correctly in the WinCC_OA.qhc collection file, we can view the results from the Qt Assistant, by opening the WinCC OA help. In general, the first noticeable result is the fact that both sets of documentation can now be viewed from the same system. Most importantly, though, it is now possible to make use of the tools that are built in the system.

4.1 Table of Contents

The first one of the tool-windows of the Qt Assistant is the table of contents, which is something that was not available before. This window presents a table of contents, implemented as a tree view for the documentation that is available. That way it is possible for the user to navigate faster and easier through the documentation.

4.2 Index

The index window is used to look up key words or phrases. In this implementation, the index directory consists of the names of the functions in the documentation. However, this does not mean that this functionality is limited to functions. On the contrary, by defining keywords in the qhp file of a manual, we can basically create an index for any word.

4.3 Search

Qt Assistant also provides a powerful full text search engine for finding specific words in the documentation. There is also the option of using the Advanced Search tool, which gives a lot more flexibility and options. For example, the user can specify some words so that hits containing these are excluded from the results, search for an exact phrase. Another option is the search for similar words. In this case the results will contain all the documents with titles similar to the search text.

In general, the list of the documents found is ordered according to the number of occurrences of the search text which they contain, with those containing the highest number of occurrences appearing first. In this case, where we have an extensive manual this order of results might not be optimal, as the user might also get irrelevant pages as top results. Therefore, it is not recommended to use the free text search for very specific terms, such as function names. A better approach would be to create keywords for these types of terms, rather than using the free text search.

4.4 Filters

With the tools provided by Qt help, it is possible to add any kind of documentation in the Qt Assistant, as long as it is in Qt compressed help files format. The JCOP documentation in particular can include different manuals for each different version of a component. For example we could register fwTrending_5.2.0.qch and fwTrending_7.0.0.qch in the WinCC_OA collection. While this practice is very convenient, it can make things complicated when performing tasks like searching for keywords, because the two manuals will have many similar keywords defined.

Furthermore, sometimes the developer might need to see the documentation for a specific project. This means that only the help for the components installed in that project should be displayed.

In order to solve both of these issues, we can selectively show the documentation, depending either on the version of a component, or on the name of a project. The custom filters and the definition of the filters attribute are already in the documentation itself and the qhp file.

4.5 GEDI Script Editor

In the GEDI script editor it has been possible to access the help of any CTRL function, simply by right clicking on its name and selecting the option to “Open Help”. Now with the use of keywords and the definition of their id attributes, it is possible to do the same for the JCOP functions. Not only that, but in theory it would also be possible to do that for any word, as long as it is defined as a keyword in the Qt help project file.

5 Conclusion

With all of these contributions, it is now possible for the developers to use a much more decent help system with a lot more functionalities than was available before. For example, it is now much easier to browse through the documentation, get faster and more reliable search results and access the help in general, even from the GEDI environment. Not only that, but the system is so flexible that it will be very easy for the future users to configure it, even locally on one machine, so that it feeds their specific needs. It is therefore evident that such changes could have a positive impact in the development process.

6 Future Challenges - Suggestions

6.1 Tool for the generation of the documentation

The process for generating the html and the Qt help files, as described above, is complicated and involves different set of tools and intermediate steps. One of the most significant issues was the fact that Doxygen is able to generate manual in the format of HTML pages while also providing many options for configuring the appearance, but when it comes to Qt help files, it provides very limited flexibility. This is a noticeable issue in this case, especially when it comes to defining filters or specifying the keywords and their attributes in the .qhp files. This happens because Doxygen has a default way of creating the qhp file which is difficult to override. Therefore, the qhp files need manual editing in order to achieve the desired result.

In the light of these issues, it would be easier if there was a tool that would edit the doxygen-generated qhp files in the appropriate way, as it is described in previous chapters. It would be even better if that tool had configuration options, so that the output could be customized depending on the purpose of the task.

Finally, assuming that such a tool will exist in the future, another idea to consider would be to attach it to Doxygen, so that the end user does not need to use both tools one after the other. For example, instead of having to run two different tools, the user could just need to run the new tool, which would run the necessary doxygen commands for him. Alternatively, we could add functions that would create the HTML documentation files, without the need of Doxygen. In both cases, the user does not need to use two different tools, one after the other, or edit the Qt files manually and the documentation can be generated in both formats (Qt and HTML) in one go.

6.2 Define panel templates

Currently, the documentation for the control panels (which are XML-based) is being displayed in manually-written HTML pages that are locally installed in the directory of its relevant component. Therefore, one idea would be to include the documentation for these panels into the current system of the unified help.

Of course, in order to do that, we need not only to create the Qt files, but also to define templates, so that XML comments would be recognizable either by Doxygen or by any similar new tool that could be used.

6.3 Integration with the fwInstallation component

As it has been mentioned before, each WinCC OA based project needs certain JCOP components, which can be added to the project with the use of the fwInstallation panel.

However, in order to achieve certain of the help-related functionalities that were described above, we need the following actions during the installation. First of all, a custom filter should be created, along with the name of the project and a relevant attribute, e.g. ProjectA and projA. Then, this attribute should be assigned to the qhp file of the component that is being installed.

In order to implement something for this purpose, it is necessary to include a function that would access and edit the component documentation and then recreate the Qt compressed help file and add it to the WinCC OA collection. On the other hand, when deleting a component from a project, the relevant attribute should be removed from the documentation.

A solution to this, would be to add a postInstall and postDelete script that would do these things for each components. Alternatively, we could add a UI button with a similar backend functionality in the fwInstallation panel, that could ask the user for the installation of help for this particular component.