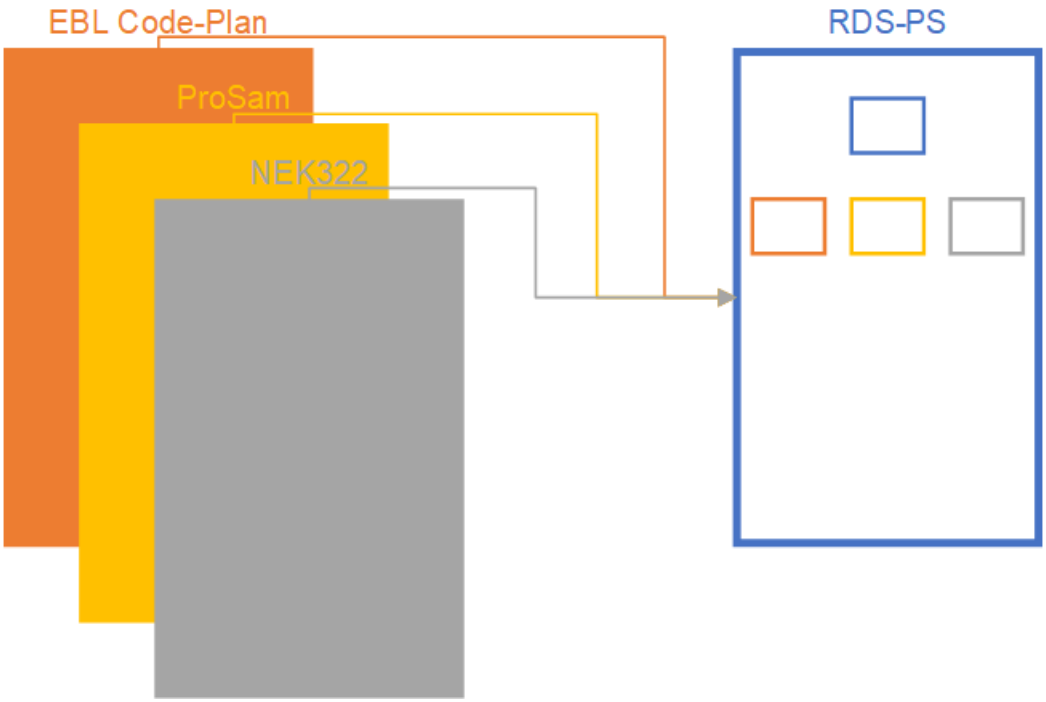
University of South-Eastern Norway

www.usn.no

FM4017 Project 2021

Evaluating the RDS-PS standard for data structures in the hydropower industry



MP-29-21

Faculty of Technology, Natural sciences and Maritime Sciences
Campus Porsgrunn

University of South-Eastern Norway

www.usn.no

Course: FM4017 Project, 2021

Title: Evaluating the RDS-PS standard for data structures in the hydropower industry

This report forms part of the basis for assessing the student's performance in the course.

Project group: MP-29-21

Group participants: Geir Lie

Supervisor: Hans-Petter Halvorsen

Project partner: **Skagerak Kraft**

Summary:

An upcoming revision of the IEC/ISO 81346-10 standard is aiming to replace existing tag standards, in the hydropower industry. Implementing the new standard is a resource demanding task. Translating existing tags to the new standard can potentially reduce the time and cost expenditure. Existing tag structures were evaluated for automatic translation. Rulesets and mappings were developed for converting the relevant structures. A Python script was written to perform the conversion. Existing tags in the NEK322 standard were unsuited for conversion to the new standard, while EBL Code-plan and ProSam tags were. 28% of the existing tags were converted to the new standard.

Preface

This report was written to fulfill the requirement for the subject FM4017 Project, in the Industrial IT and Automation master program at University of South-Eastern Norway. The project itself was performed in collaboration with the project partner at their office in Porsgrunn. The student was employed in a part-time position at the company, but the project was not performed in that capacity. Appendix A contains the task description. Kickoff for the project was 27. August 2021 and the report deadline 19. November 2021. The project schedule is contained in Appendix B.

The report is intended for readers assessing the performance of the student in the subject. It might also be of interest to persons working with implementing RDS-PS in their companies.

I would like to thank my coworkers involved in the project. They provided invaluable information and helped me gain experience with project management.

Porsgrunn, 18.11.2021

Geir Lie

Contents

Nomenclature

CSV – Comma-separated values

CW – Construction Works

EBL – The Norwegian Electricity Industry Association

IEC – International Electrotechnical Commission

ISO – International Organization for Standardization

NEK – Norwegian Electrotechnical Committee

PS – Power Supply

RDS – Reference Designation System

SCADA – Supervisory control and data acquisition

1 Introduction

An upcoming revision of the IEC/ISO 81346-10 standard, referred to in this report as RDS-PS, is targeting power supply system. This revision is relevant for the Norwegian hydropower industry. It aims to addresses problems of existing systems, used to designate components with tags, and facilitate digitalization [1]. Skagerak Kraft is therefore interested in adopting this new standard. Implementing it represents a substantial amount of work. The goal of this project was to utilize existing data structures to automatically create corresponding RDS-PS tags. It was also a sub-goal to identify new use cases, enabled by the new structure, in the Operations Portal. This included new filtering options and a corresponding test plan for development. The Operations Portal is an internal web service for the maintenance personnel.

The existing structures considered in this project consisted of EBL code-plan, used in the maintenance system, NEK322, used in vendor documentation, and ProSam used in the SCADA system. They were evaluated for consistencies between power plants and relevance to RDS-PS. Tags without relevance to part 10 of the standard were excluded. Methods for conversion, with a minimal number of mapping rules, were developed based on the evaluation. Mapping rules were created for converting from the existing standards to RDS-PS. A script to convert the existing tags was created in Python [2]. The functional aspect of RDS-PS was the only one considered. A system sketch of the conversion process can be seen in Figure 1.

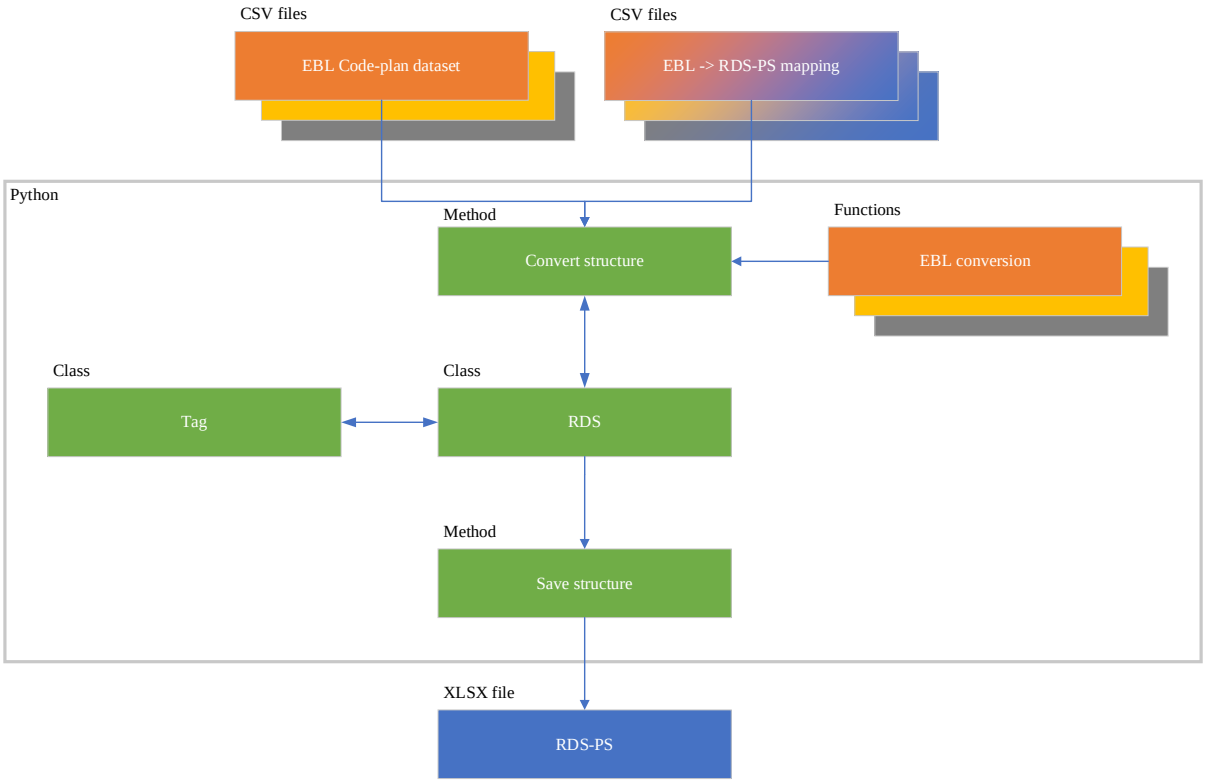


Figure 1: Conversion process

1.1 RDS-PS

ISO/IEC 81346 “Industrial systems, installations and equipment and industrial products - Structuring principles and reference designations” describes how to formulate unambiguous references to objects in any system [3]. These objects can be represented in several different *aspects*. Each aspect focuses on providing information in a different way. The functional aspect, for example, describes the system with respect to the functions that exists, while the location aspects give information about the physical placement. A symbol in the tag gives information about which aspect it belongs to [4]. The equality symbol is used to indicate the functional aspect. An example of a complete tag is given in Figure 2.

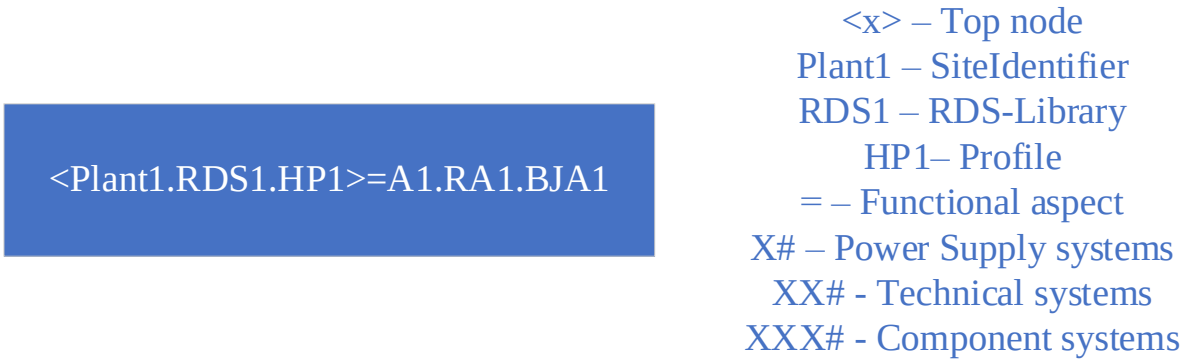


Figure 2: RDS tag in the functional aspect [4]

The tag is made up of a top node and a combination of class codes. Each class code consists of either one, two or three letters together with a serial number. The serial number is not intended to convey any information, other than to distinguish between multiple occurrences of the same class. One letter class codes represent top level objects while the three letter classes are the most detailed. The tag can consist of any number of class codes, in any order, which best describe the system. It is required that a one letter class code comes directly after the top node. When part 12, Construction Works, is used together with part 10, it is recommended to deviate from this and use a two-letter class following the top node. The top node is required to contain a unique identifier, which part of the standard and which version was used to create the structure. It can contain additional information if it is deemed beneficial.

1.2 Existing tag structures

The EBL code-plan was distributed in the form of an Excel file [5]. This file describes tags with up to 5 levels. Each level has an integer value between 0 and 999. It is augmented with 3 higher levels in Skagerak Kraft’s maintenance system. The first level describes which company, and the following levels are for area group, area, group, unit, component, part component and part. The level for company and area group was ignored in this project. Area group is used to group several plants and waterways, based on physical location. An example tag can be seen in Figure 3.

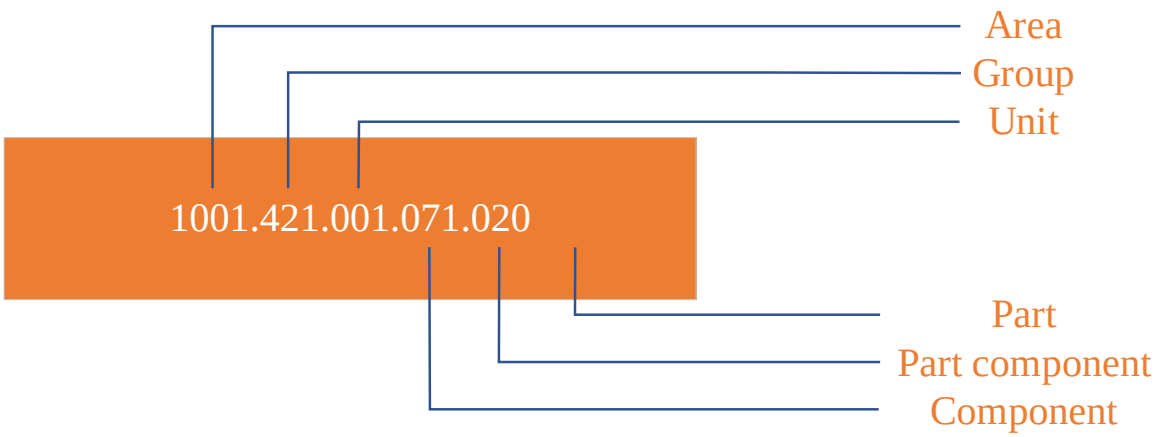


Figure 3: EBL code-plan tag

Area is used to reference a specific powerplant or waterway. The following level is an EBL class code. These classes cover the largest structures, such as control system, turbine, reservoir, and buildings. Unit is used to number repeating occurrences of the same group class, within an area. A zero unit is used to hold general structures which are independent of multiple occurrences. The component level describes different objects the group consists of. For a turbine, this contains runner, draft tube and bearing, among others. Some of the numbers refers to a single possible occurrence of a part, such as the draft tube, while others have a dedicated series of numbers. A turbine bearing is an example such a component, where a range of numbers contains other component related to this bearing. There is a gap in the numbering of the next non-bearing component, which allows for having several bearings. Part component is more detailed than component and have a similar numbering schema, where some numbers identify unique occurrences and others have a dedicated repeatable series. The part level is not used in Skagerak Kraft. Application of the EBL code-plan within the company is deliberately restricted in detail. Most larger components are not described to the detail the standard facilitates.

NEK 322, “Documentation Used in Electrotechnology – Part 2: Power Supply Systems”, is withdrawn [6]. It is still in use for new projects and current documentation. The standard is a part of a series, ranging from NEK321 to NEK325 [7]. NEK321 covers the general requirements, while the others address specific industries. It defines tags for the individual physical components, and larger units, in a power plant. This standard series is used throughout the documentation of the different plants. Cables, relays, and terminal blocks are examples of individual products which have references with this system in the documentation. Unfortunately, the storage format of the documentation for the different plan is inhomogeneous. Older installations are documented with scanned copies of original paper, while newer are in PDF or even contains component lists in spreadsheets. In addition, there is a large variation in the structure of the documents from different suppliers. An example of a tag can be seen in Figure 4.

Figure 4: NEK322 tag

The tag described by the standard can belong to different groups, which convey different information. Three groups, with identifiers, are specifically described: Function-oriented (=), product-oriented (-) and placement-oriented (+) structure. The example contains the corresponding tag from each group. Each tag can contain several levels. The level contains a class code and a serial number. No information is attributed to the serial number, it only serves to separate multiple occurrences of the same class within the same level. The different groups are sometimes combined, in existing documentation, to provide the information in a single line of text.

ProSam, "Produksjon med samarbeidende driftssentraler" (translated: Production with Cooperating Operation Centrals), was implemented by Statkraft in 2006 [8]. It defines a tag structure for communicating with their operation central. The tag structure is not further explained in this report, in order to present the work with open access.

1.3 Relationships between the new and old tags

The use of EBL and NEK322 in Skagerak Kraft's systems targets the same type of objects. They are used to identify physical components in the plants. The main difference in application, is the level of detail. This supports the intended functions they perform in the company. In addition, NEK322 is limited to electrotechnical components. EBL is intended to be used for all technical disciplines. There is also a difference in the amount of information contained. EBL focuses solely on the existence of the object, while NEK322 contain information about the placement, functionality, and which component the object is a part of. ProSam focuses on different objects, compared to the other existing structures. It is used to describe data exchange in the SCADA system. This means that it contains information about sensors and events in the power plants, which constitutes the functionalities of the physical objects.

RDS-PS is intended to replace all of the tag systems considered in this project [1]. This alludes to a relationship between the existing systems and RDS-PS. NEK322 has the strongest resemblance, both with respect to the information contained and the tag structure itself. Both RDS-PS and NEK322 describes multiple structures, each intended to convey different information. The structures in NEK322 have similar counterparts in RDS-PS and uses the same symbols to distinguish between them. This indicates the possibility of automatic conversion. EBL contains information for components that are not present in the other existing systems. It also has a rigid structure, which should help with mapping it to RDS-PS. ProSam also has information to contribute, which is not present in the other systems. This relates mainly to functionalities of the controls systems. The functional aspect of RDS-PS is suited to contain this information.

2 Methods

The existing data structures were extracted from their systems. Microsoft Excel [9] was used to validate the consistency within each structure and if they held information relevant to the functional aspect of RDS-PS. Excel was also used to create a minimal representation of the different tags in each system. This representation was used to create a mapping between each structure and RDS-PS. Python was used to implement an automatic conversion between the existing data and a new RDS-PS structure.

2.1 Validation of existing data

Documentation for the control systems was targeted to be used as a source for creating a dataset of known NEK322 designated components. The different EBL tags were available in the maintenance system as a dataset. A dataset of ProSam tags was exported from the SCADA system.

The content of the columns in the EBL dataset was cleaned with a Power Query in Excel. Tags which had a waterway as the area was removed. This was because information about relationships to power plants, was not contained in the tag. Columns which did not contain information relevant to the conversion was removed. This included the columns for company, area group and part. The query also separated the EBL class from the description and combined all the columns into one tag. A filter was also implemented to remove EBL groups not relevant to RDS-PS. The query can be seen in Appendix C and the groups that were filtered out in Table 1.

The consistency of the EBL dataset was validated by grouping the tags with a Power Query. A new column was created to group the tags by. The column had the unit class replaced with an exclamation mark. In addition, the area class was removed from tags that had additional classes. The description for each row in the group was then concatenated and added as a new column. A column was also added with the count of rows within each group. Each group was then inspected to verify that the descriptions of the tags within were consistent with the code-plan. This was a prerequisite if the tag code could be used to automatically be translated. The Power Query for grouping the tags can be seen in Appendix D.

Table 1: EBL classes filtered out (Norwegian names) [5]

Group	Name
885	LØFTE- OG HEISEINNRETNINGER
891	VERNE- OG SIKKERHETSUTSTYR (FLYTTBART)
000	Felles
894	MÅLEINSTRUMENTER (TRANSPORTABLE)

890	DIVERSE
881	KJØRETØY OG BÅTER
877	GRUSTAK, TIPP OG DEPONI
876	GRØNTANLEGG-PARKER-PLASSER
871	VEIER OG BRUER
870	UTEANLEGG
855	BÅTHUS
853	HYTTER
847	AGGREGAT- OG SPILLHUS
877	GRUSTAK, TIPP OG DEPONI
846	FJELLROM ADKOMSTTUNELLER
845	VERKSTED OG LAGER
841	STASJONSBYGG I FJELL
842	PORTALBYGG
843	STASJONSBYGG UTE
878	VANNFORSYNING
830	ADMINISTRASJONSBYGG - VELFERDSBYGG
831	ADMINISTRASJONSBYGG

A similar method was used with the ProSam dataset. The number of ProSam tags were several times larger than EBL. The dataset was therefore evaluated by grouping the tags independent of the first and last part of the tag. The number of different power plants represented in each group was then counted. Tag groups which contained less than 10% of the power plants, were filtered out.

2.2 Translation of existing data

The validated EBL dataset was used as the basis for creating a mapping to RDS-PS. Each row from the dataset was used as an entry in the tag mapping. The mapping also contained the columns with concatenated tag descriptions and count, to guide the effort when mapping it to RDS-PS. An additional column was added to contain the corresponding RDS-PS tag. A second column was added, which contained a number to identify if the row was a mapping of the area part of the tag or the remainder. The symbol "X" was used in the RDS-PS tag for EBL tags without a clear conversion. This was relevant for groups containing tags with different functionality. The symbol "!" was used in the RDS-PS if the unit number was necessary for the conversion. Mapping of some example tags are showcased in Figure 5.

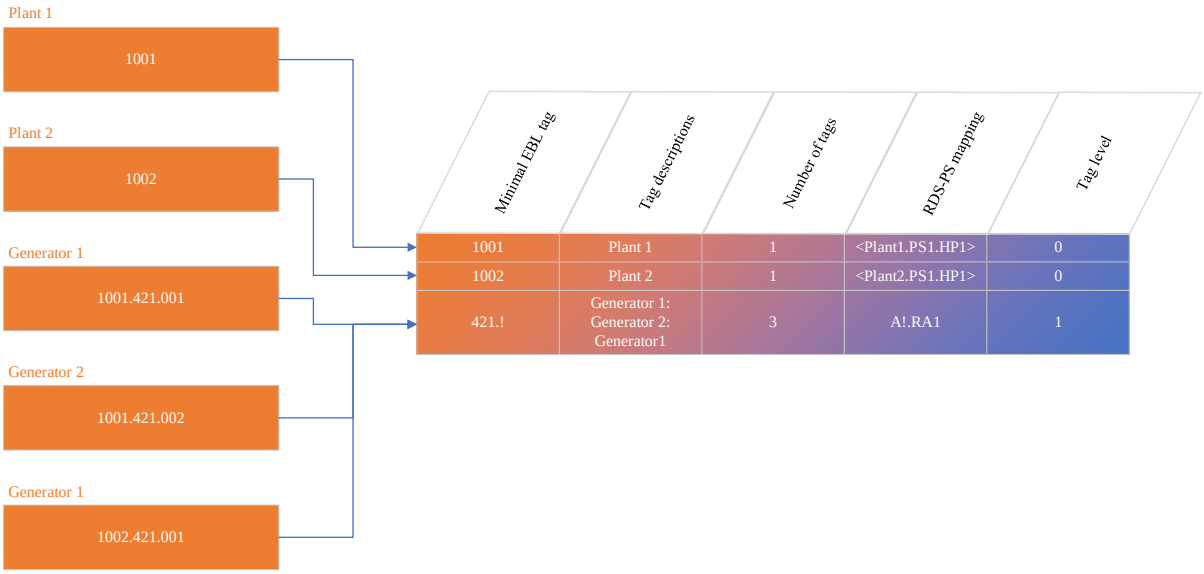


Figure 5: Illustration of the mapping process

Some combinations of static numbers were used to confer special meaning to tags. The number 1 was used as the serial number for classes which were known to occur once. The class "JB" with number 5 was used to identify a common waterway. "H2" was used to refer to a bailing disposing system. The high voltage tags were mapped to a combination of the class "B" and the voltage level. "D1" was reserved for the low voltage support system and "D2" for the direct current support system. "F1" was used for tags mapping to the managing system.

A similar approach to the EBL tags was used for the ProSam dataset. The tags from the original dataset were split into three parts and separated into different tables. Repeating occurrences of substrings were removed from the tags. The split occurred after the first part and before the last part in the tag. String manipulation was performed on the middle part to replace identifying numbers with an "!" symbol. Each table was then grouped by the tag. A concatenated column was added, with the descriptions, and a column with the count of rows. The table with the middle part of the tag was filtered to only contain groups that had occurrences in at least 10% of the plants. The table with the last part was filtered to only contain groups with more than 10 tags. A union was then performed on the three tables to create a mapping file for ProSam tags, with the same structure as for EBL. The column which

indicated which part of the tag it mapped, contained 0 for the first part, 1 for the second and 2 for the final part.

The same reserved classes were reused from the EBL mapping. A new symbol, "|", was introduced to indicate voltage level in the ProSam tag. This symbol was then used in the mapping to correspond to a "B|" class in RDS-PS.

2.3 Automatic creation of RDS-PS tags

A Python script was chosen as the tool for automatic conversion of the existing tags. The analysis and design of the software was performed in a simple fashion, without making use of a formal development process. It consisted of identifying the necessary classes and methods, in order to convert tags based on a provided dataset, mapping and conversion function. The result of conversion from each data source was to be added to an RDS-PS structure, which could then be printed to the screen or saved as an XLSX file.

The code was structured with two classes. A class named "RDS" for controlling the conversion process and a "Tag" class for storing tag data and methods. A tree structure was used to organize the tags. Conversion from the original standards was handled with a method in the "RDS" class, which accepted a function in addition to the tag file and corresponding mapping file. A function for conversion was created for each relevant standard. These functions were created to handle the differences between the standards. The usage of the created classes can be seen in Table 2.

Table 2: Python code usage example

```
rds = RDS()
rds.convert_structure("path/EBL_dataset.csv", "path/EBL_mapping.csv",";",
convert_EBL)
rds.convert_structure("path/ProSam_dataset.csv",
"path/ProSam_mapping.csv",";", convert_ProSam)
rds.print()
```

The EBL function splits the tag into two, with the first part containing the power plant and the second containing the structure. The corresponding function for ProSam splits the tag into three sub-tags. They both perform string manipulation, afterwards, to look for a mapping for each sub-tag. Figure 6 illustrates the conversion function for EBL tags. These functions accepted lists with the dataset and corresponding mapping. They returned a list of translated RDS-PS tags, a list of the original tags and a string containing the source of the tags. The data was then added to the RDS-PS structure by recursively searching through the structure. Dummy objects were created whenever it was not explicitly specified by the converted tags. The name of the dummy object was added if it was defined by another tag later.

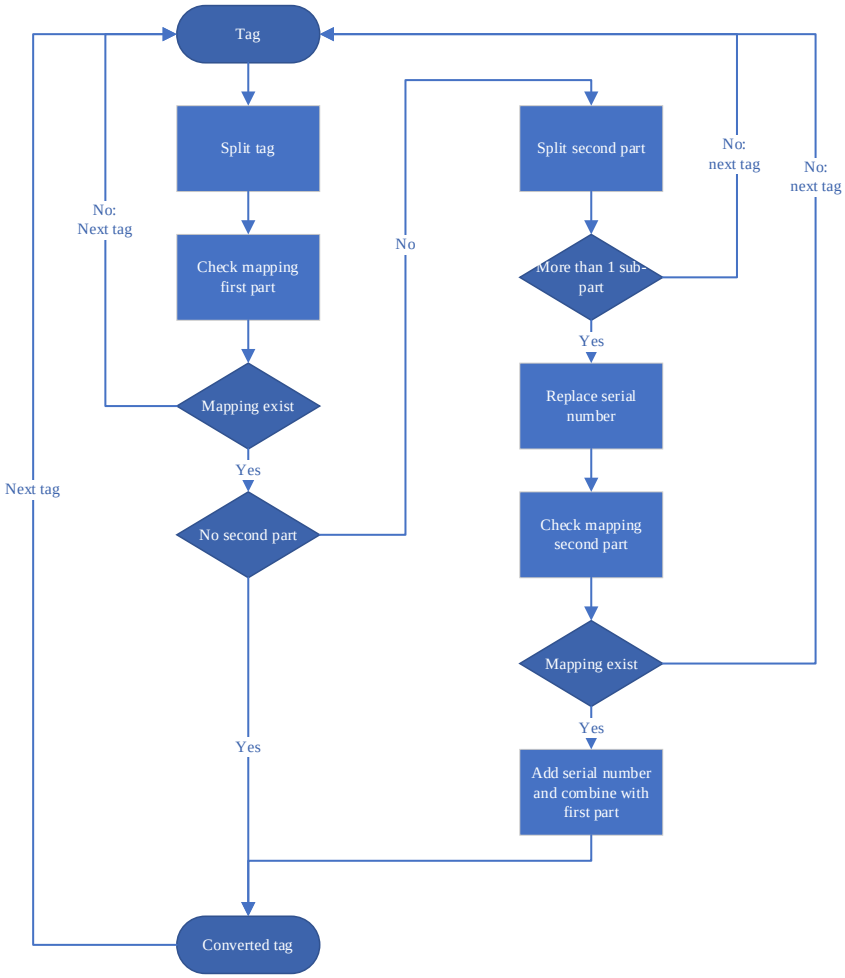


Figure 6: EBL conversion function flowchart

A method in the “RDS” class was made to export the converted structure. It had an XLSX file, with a row for each RDS-PS tag, as output. The “pandas” library was used to create the file [10]. It had columns for storing the complete RDS-PS tag, the descriptive name and a column containing the original tags which corresponds to it. The last column was structured as the string representation of a Python dictionary, with a key for each of the old systems, which contained a list of the relevant original tags. The source code for the script is contained in Appendix F.

The functionality of the script was verified by using the “unittest” framework, from the Python standard library. Only unit tests were created, because of the simple relationship between the created classes. 11 tests were created for the “Tag” class, 6 for the “RDS” class, 8 for the conversion function for EBL and 5 for the conversion function for ProSam. The tests can be seen in Appendix G.

3 Result

A dataset of NEK322 tags was not created. The different EBL tags were available in the maintenance system as a dataset, consisting of 3380 tags. A dataset of 11667 different ProSam tags was gathered from the SCADA system.

The EBL dataset was consistent, independent of power plant. A total of 2352 tags from the dataset were designated as relevant for RDS-PS. The ProSam dataset contained tags which were independent of power plant. A total of 199 groups, with 7764 tags combined, contained a structure which were present in at least 10% of the plants. Based on the validation, was EBL and ProSam both considered possible and useful to convert into RDS-PS.

The created EBL mapping consisted of 274 rows in a CSV file, mapping two different parts of the tag. An anonymized version of the mapping can be seen in Appendix E. Corresponding RDS-PS tags was found for 78 of the rows. The total mapping of ProSam tags consisted of 4779 rows, representing three different parts of the tag. This number was filtered down to 422 rows. A translation into RDS-PS was identified for 335 of the rows.

The conversion script passed all the tests. With the mappings created, it was able to convert 1090 of the EBL tags. The corresponding number for ProSam tags was 2897. An XLSX file with RDS-PS tags, and which tags they originated from, was the result. The total percentage of converted relevant tags was 28%.

4 Discussion

The revision of the standard was not released before the project ended. As a result, it is necessary to consider the validity of the content in this report as subject to change.

Creating a dataset with tags described with NEK322 could have been done. The existing documentation makes extended use of it. It was deemed to be out of the scope to create a dataset, due to the varying file formats and PDF structures. This limited the result from the conversion. NEK322 is probably the existing standard which most closely resembles RDS-PS. Creating a mapping for it might therefore be more efficient than for the other structures. It is also the existing structure with the most relevance for other aspects of RDS-PS. Since no dataset was created, was it impossible to check how consistently the standard is applied. There might be inconsistencies in the way it is used on different plants, which could make it impossible to translate with the method developed in this project.

EBL tags with waterway area codes were ignored in this project. The process of creating relationships between them and the powerplants was estimated to require more resources than were available in the project. There was more information available about the waterways in the EBL dataset than the ProSam. The omission of the tags from the EBL dataset might have prevented the discovery of problems with the conversion method. Other tags were also excluded, based on the group code. The filter used impacted the created mapping. Choosing a different boundary between PS and CW, might also expose problems with the methods used in this project.

Information was lost when the ProSam dataset was reduced, to only those categories existing in multiple plant. This approach drastically decreased the number of entries in the mapping, but it might suppress problems with the general approach used. No verification was done to check which tags were removed from the dataset. The reduction was necessary to complete the mapping with the allocated resources. It also helped to focus the translation, on the most general tags. The tags in the dataset were not evaluated for relevance to RDS-PS. This affects the percentage of tags converted.

A separate mapping was created for each of the data structures translated in this project. Keeping both updated, with respect to each other, might reduce the advantage of using the automatic conversion script. This challenge is amplified by the uncertainty regarding how to apply the new standard, within the project group. Using serial numbers without meaning was ignored in this project. The conversion script can be extended to trim down the numbers in the converted structure, but it is helpful to clearly see the reserved numbers while the RDS-PS is still in an early face of implementation in the company.

There was also a large uncertainty regarding verifying the converted tags. The revision of the ISO standard was not released during the project and no generally accepted structure was found to compare against. As a result, there might be problems converting tags that was not discovered in this project. The resulting conversion method developed in this project may be useful for Skagerak in the future. As new features are added to the maintenance system, including support for RDS-PS, will it be necessary to document the relationship between the tag structures. The automatic conversion, based on the mappings, can provide this.

Several problematic components were discovered. Station transformers are likely to be untranslatable based on the EBL or ProSam tag. No information was found in the relevant

tags about what function it performs. The same goes for other parts of the high voltage system. It is expected that such systems should be divided based on galvanic separation, according to RDS-PS. There are few tags for busbars or cables in the datasets considered in this project. Translating the high voltage system in general might therefore be impossible with the developed method. Breakers and switches also represent a problem, as there needs to be a heavy emphasis on reserved tags to combine them from both existing structures. Considering the discovered problems with the method, along with the possibilities of undiscovered problems, is there a likelihood that there is little time saved by automatically converting existing tag structures.

Focusing on the existing systems may hamper acclimation to the new standard. Many of the translated tags are general to any power plant. A better way to reduce the work with implementing RDS-PS might be to create templates for different configurations used in power plants. This would be conducive to understanding the standard, while reducing repetitive work. Basing the templating system on common configurations would also limit the amount of work required if there was a change in the application of the standard.

The sub-goal of implementing new features in the Operation Portal was not achieved. Priorities in the development of the service was focused elsewhere and it was not possible to schedule it during the project. This also included the discovery of use cases, which also made it impossible to create a test plan for new filtering options.

5 Further Work

The approach used in this project for RDS-PS, should be transferable to RDS-CW. Only a fraction of the tags described with ProSam are likely to be of relevance, but several classes filtered out from EBL are.

Each of the existing tag system contains different information about the power plants. NEK322 contains the most detail and is the only which provides information about the physical placement of components. Creating a dataset of NEK322 tags should be a priority if a similar approach is to be tried with the location aspect. It will likely also improve the conversion for the functional aspect.

It is recommended to revisit the result from this project after the revision to the RDS-PS standard is released, along with guidelines for the reference designation process. The mappings will need to be updated, as the target structure changes. A method to incorporate EBL tags for the waterways should be devised in conjunction with this. It is also recommended to compare the created RDS-PS tags to other companies in the industry, as a method to verify the results and harmonize the application of the standard.

6 Conclusion

Components tagged with NEK322 was found unusable for conversion to RDS-PS in this project, due to the format the relevant documentation exists in. The systems using EBL and ProSam contained accessible and structured information about the power plants. 1090 relevant EBL tags out of 2352 were converted into RDS-PS, and 2897 out of 11667 ProSam tags were converted. No functionality for filtering, or other utilization for RDS-PS, was implemented in the Operation Portal.

References

[1] Energi Norge, "Digitalisering: RDS-Hydro Power," [Online]. Available: https://www.energinorge.no/fagomrader/forskning/forskningsprosjekter/produksjon/RDS-Hydro-Power/. [Accessed 15 11 2021].

[2] The Python Software Foundation, "The Python Language Reference," The Python Software Foundation, 18 12 2019. [Online]. Available: https://docs.python.org/release/3.8.1/reference/index.html. [Accessed 14 11 2021].

[3] ISO, "IEC 81346-1:2009 - Industrial systems, installations and equipment and industrial products — Structuring principles and reference designations — Part 1: Basic rules," [Online]. Available: https://www.iso.org/standard/50857.html. [Accessed 04 10 2021].

[4] Energi Norge, "Introduksjon til RDS Power systems," 2021. [Online]. Available: https://www.energinorge.no/kurs-og-konferanser/2021/01/rds-isoiec-81346-referansestruktur-for-digitalisering-i-vannkraften/. [Accessed 14 11 2021].

[5] Energibedriftenes Landsforening (Energi Norge), "Publication 54-2001," Energi Norge, 2001.

[6] Standard Norge, "NEK 322:1995," Standard Norge, 01 03 2019. [Online]. Available: https://www.standard.no/en/webshop/productcatalog/productpresentation/?ProductID=131870. [Accessed 14 11 2021].

[7] Norsk Elektroteknisk Komite, Documentation used in electrotechnology, part1: general requirements, Oslo: Standard Online AS, 1999.

[8] K. Strøm, "Driftssentraler samordnes," Teknisk Ukeblad, 23 05 2006. [Online]. Available: https://www.tu.no/artikler/driftssentraler-samordnes/325957. [Accessed 04 10 2021].

[9] Microsoft, "Microsoft Excel (365)," 2021. [Online]. Available: https://www.microsoft.com/en-us/microsoft-365/excel. [Accessed 14 11 2021].

[10] The pandas development team, "pandas-dev/pandas: Pandas," Zenodo, February 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3509134. [Accessed 14 11 2021].

Appendices

Appendix A Task description

Appendix B Project schedule

Appendix C Power Query for filtering and cleaning the EBL dataset

Appendix D Power Query for validating the EBL dataset

Appendix E EBL to RDS-PS mapping

Appendix F Python conversion script source code

Appendix G Python test scripts

Appendix A Task description

FM4017 Project

Title: Evaluating the RDS standard for data structures in the hydropower industry

USN supervisor: Hans Petter Halvorsen, Nils-Olav Skeie

External partner: Skagerak Kraft AS

Task background:
The Reference Designation System – Hydro Power (RDS-PS) standard, part 10 of IEC/ISO 81346, is expected to launch in 2021. This standard is to be implemented in the industry. There are currently several different structures used to describe hydropower plants, for different use cases. For example, a structure might be used primarily for maintenance history, SCADA, or schematics. This creates a barrier to combine data from different systems, not only within a company but also between different companies. Implementing the new RDS standard, which aims to replace the existing structures, will facilitate new ways to leverage the existing data.

Task description:

- Make an overview of existing data structures like NEK322, PROSAM and EBL that are used to describe power plants and identify to which degree they comply between the plants. The focus should be PROSAM and EBL.
- Include a description of the RDS-PS standard and relate the description to the overview of existing data structures.
- Compare the existing data structures and create an overview of existing components that can be useful for the RDS-PS standard.
- Analyse and design a software application in Python that, to the largest extent feasible, can automate the creation of the new RDS structure. Among the different aspects available in RDS, will the functional aspect be prioritized.
- Identify the usefulness of implementing the new structure. This will be explored by using the structure to implement new features in the Operation Portal (Service developed by the external partner).
- Prepare the RDS structure to be used as a filtering option in the Operation Portal.
- Make a test plan for testing the filtering feature.

Student category: IIA industry master students employed at Skagerak Kraft AS

The task is suitable for students not present at the campus (e.g. online students): No

Practical arrangements:

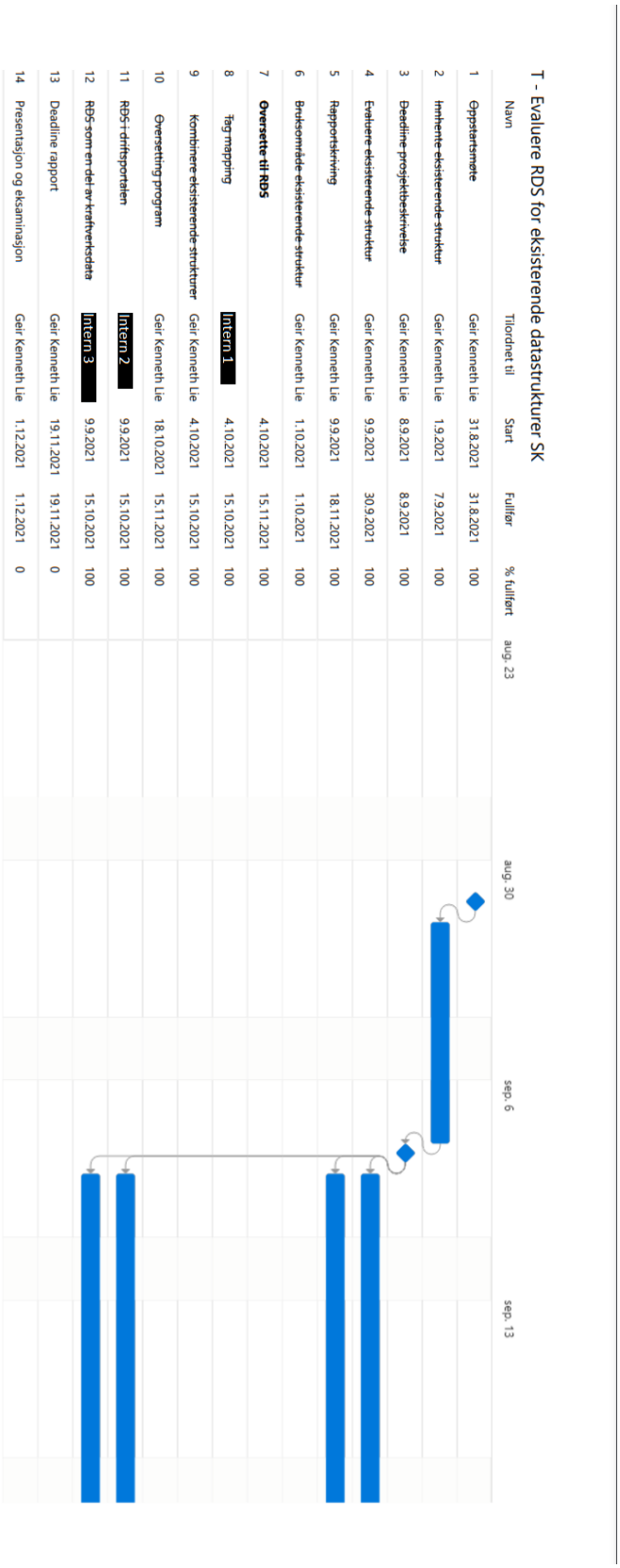
Project group led by the student with participants from the external partner. The participants contribute with domain specific knowledge in power plant structure, RDS and existing data systems.

Signatures:

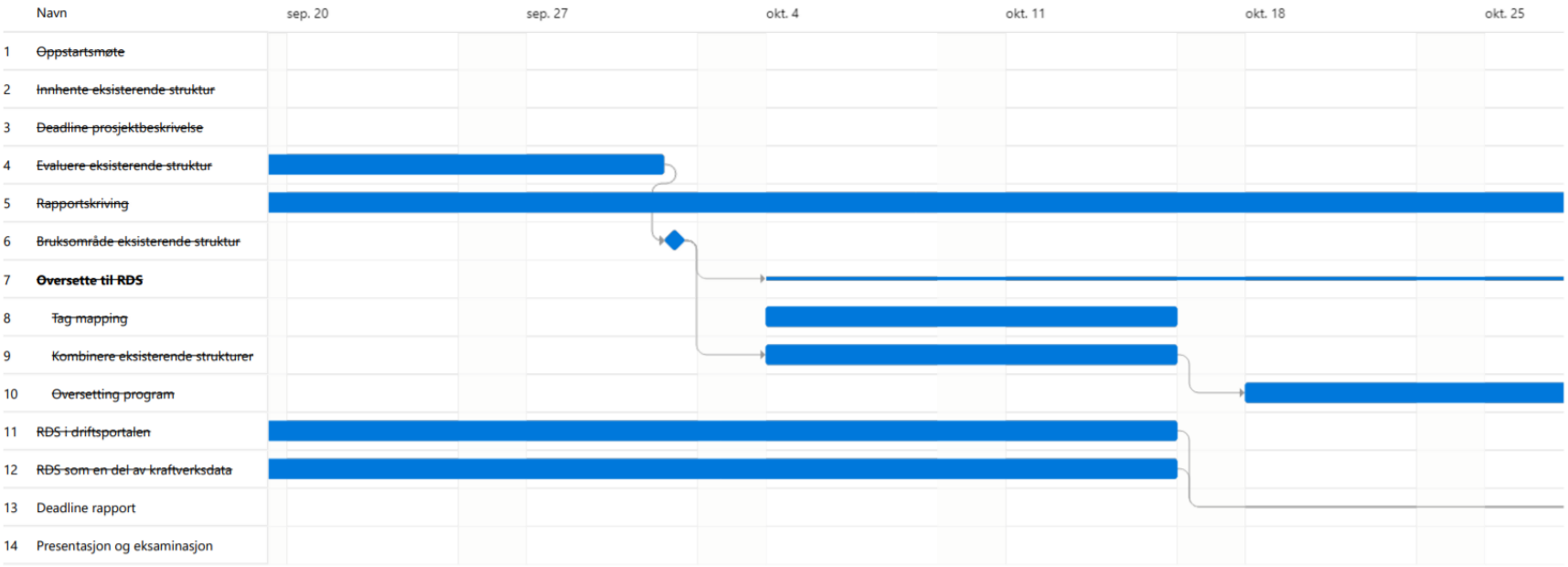
Supervisor (date and signature):

Students (write clearly in all capitalized letters + date and signature):

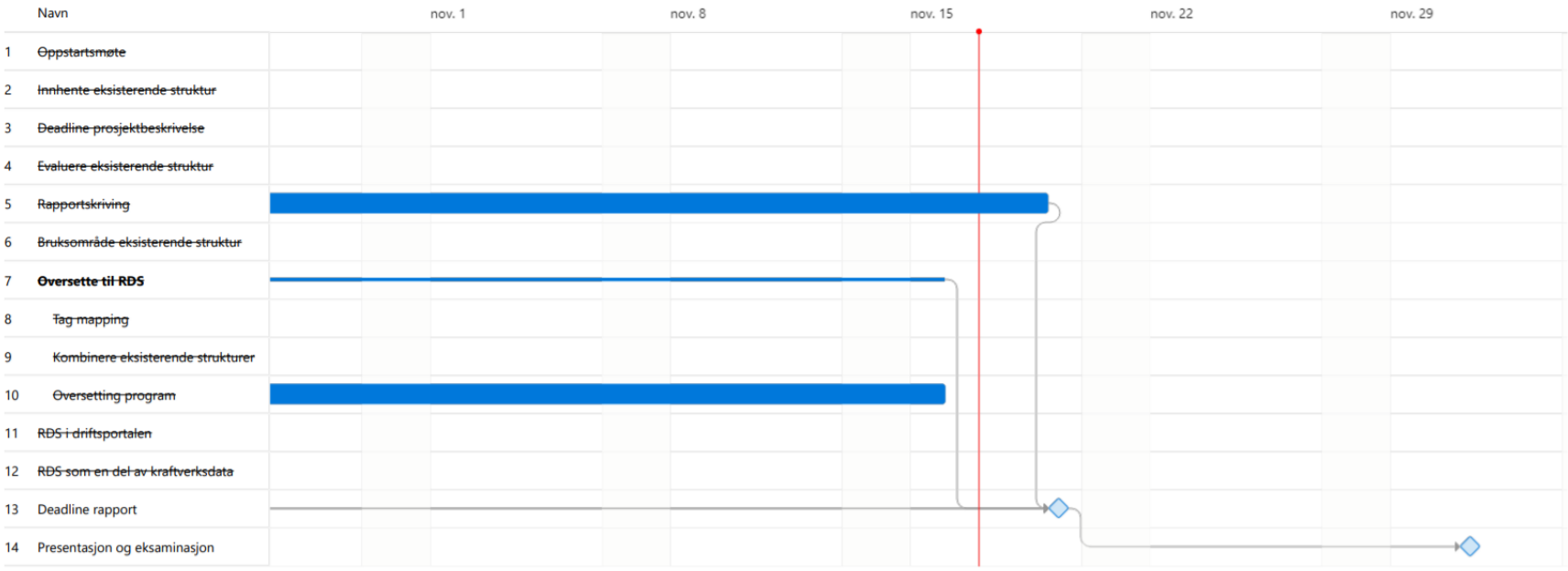
Appendix B Project schedule



T - Evaluere RDS for eksisterende datastrukturer SK



T - Evaluere RDS for eksisterende datastrukturer SK



Appendix C Power Query for filtering and cleaning the EBL dataset

```
let
    Kilde = ELB_Ra,
    #"Fjernede kolonner" = Table.RemoveColumns(Kilde,{"Del"}),
    #"Filtrerte rader" = Table.SelectRows(#"Fjernede kolonner", each [Områdegruppe] <> null and [Områdegruppe] <> ""),
    #"Fjernede kolonner1" = Table.RemoveColumns(#"Filtrerte rader",{"Selskap"}),
    #"Filtrerte rader1" = Table.SelectRows(#"Fjernede kolonner1", each [Område] <> null and [Område] <> ""),
    #"Fjernede kolonner2" = Table.RemoveColumns(#"Filtrerte rader1",{"Områdegruppe"}),
    #"Del kolonne med skilletegn" = Table.SplitColumn(#"Fjernede kolonner2", "Område", Splitter.SplitTextByEachDelimiter({" -"}, QuoteStyle.Csv, false), {"Område.1", "Område.2"}),
    #"Endret type1" = Table.TransformColumnTypes(#"Del kolonne med skilletegn",{{"Område.1", type text}, {"Område.2", type text}}),
    #"Filtrere anlegg" = Table.SelectRows(#"Endret type1", each not List.Contains(FilterAnlegg,[Område.1])),
    #"Fjernede kolonner3" = Table.RemoveColumns(#"Filtrere anlegg",{"Område.2"}),
    #"Del kolonne med skilletegn3" = Table.SplitColumn(#"Fjernede kolonner3", "Komponent", Splitter.SplitTextByEachDelimiter({" -"}, QuoteStyle.Csv, false), {"Komponent.1", "Komponent.2"}),
    #"Del kolonne med skilletegn1" = Table.SplitColumn(#"Del kolonne med skilletegn3", "Gruppe", Splitter.SplitTextByEachDelimiter({" -"}, QuoteStyle.Csv, false), {"Gruppe.1", "Gruppe.2"}),
    #"Endret type2" = Table.TransformColumnTypes(#"Del kolonne med skilletegn1",{{"Gruppe.1",  type text}, {"Gruppe.2", type text}}),
    #"Filtrere klasse" = Table.SelectRows(#"Endret type2", each not List.Contains(FilterKlasser,[Gruppe.1])),
    #"Fjernede kolonner4" = Table.RemoveColumns(#"Filtrere klasse",{"Gruppe.2"}),
    #"Del kolonne med skilletegn2" = Table.SplitColumn(#"Fjernede kolonner4", "Enhet", Splitter.SplitTextByEachDelimiter({" -"}, QuoteStyle.Csv, false), {"Enhet.1", "Enhet.2"}),
    #"Endret type3" = Table.TransformColumnTypes(#"Del kolonne med skilletegn2",{{"Enhet.1",  type text}, {"Enhet.2", type text}}),
    #"Fjernede kolonner5" = Table.RemoveColumns(#"Endret type3",{"Enhet.2"}),
    #"Endret type4" = Table.TransformColumnTypes(#"Fjernede kolonner5",{{"Komponent.1",  type text}, {"Komponent.2", type text}}),
```

```
    #"Fjernede kolonner6" = Table.RemoveColumns(#"Endret type4",{"Komponent.2"}),
    #"Del kolonne med skilletegn4" = Table.SplitColumn(#"Fjernede kolonner6",
"Delkomponent", Splitter.SplitTextByEachDelimiter({" -"}, QuoteStyle.Csv, false),
{"Delkomponent.1", "Delkomponent.2"}),
    #"Endret type5" = Table.TransformColumnTypes(#"Del kolonne med
skilletegn4",{{"Delkomponent.1",  type text}, {"Delkomponent.2", type text}}),
    #"Fjernede kolonner7" = Table.RemoveColumns(#"Endret type5",{"Delkomponent.2"}),
    Egendefinert1 = Table.AddColumn(#"Fjernede kolonner7", "EBL_Kode", each
Text.Combine(List.Select({[Område.1],[Gruppe.1],[Enhet.1],[Komponent.1],[Delkomponent
.1]},  each _<> "" and _ <> null),".")),
    #"Fjernede kolonner8" = Table.RemoveColumns(Egendefinert1,{"Område.1", "Gruppe.1",
"Enhet.1", "Komponent.1", "Delkomponent.1"}),
    #"Omorganiserte kolonner" = Table.ReorderColumns(#"Fjernede
kolonner8",{"EBL_Kode", "Navn"})
in
    #"Omorganiserte kolonner"
```

Appendix D Power Query for validating the EBL dataset

```
let
    Kilde = #"Formatere og filtrere data",
    #"Del kolonne med skilletegn" = Table.SplitColumn(Kilde, "EBL_Kode", Splitter.SplitTextByEachDelimiter({"."}, QuoteStyle.Csv, false), {"EBL_Kode.1", "EBL_Kode.2"}),
    #"Del kolonne med skilletegn1" = Table.SplitColumn(#"Del kolonne med skilletegn", "EBL_Kode.2", Splitter.SplitTextByEachDelimiter({"."}, QuoteStyle.Csv, false), {"EBL_Kode.2.1", "EBL_Kode.2.2"}),
    #"Del kolonne med skilletegn2" = Table.SplitColumn(#"Del kolonne med skilletegn1", "EBL_Kode.2.2", Splitter.SplitTextByEachDelimiter({"."}, QuoteStyle.Csv, false), {"EBL_Kode.2.2.1", "EBL_Kode.2.2.2"}),
    #"Fjernede kolonner1" = Table.RemoveColumns(#"Del kolonne med skilletegn2",{"EBL_Kode.2.2.1"}),
    #"Egendefinert lagt til" = Table.AddColumn(#"Fjernede kolonner1", "Egendefinert", each "!"),
    #"Omorganiserte kolonner" = Table.ReorderColumns(#"Egendefinert lagt til",{"EBL_Kode.2.1", "Egendefinert", "EBL_Kode.2.2.2", "Navn"}),
    #"Kolonner med nye navn" = Table.RenameColumns(#"Omorganiserte kolonner",{{"EBL_Kode.1", "1"}, {"EBL_Kode.2.1", "2"}, {"Egendefinert", "3"}, {"EBL_Kode.2.2.2", "4"}}),
    #"Egendefinert lagt til1" = Table.AddColumn(#"Kolonner med nye navn", "Egendefinert.1", each if [2] = null then [1] else if [3] = null then [2] else if [4] = null then [2]&"."&[3] else [2]&"."&[3]&"."&[4] ),
    #"Fjernede kolonner" = Table.RemoveColumns(#"Egendefinert lagt til1",{"1", "2", "3", "4"}),
    #"Omorganiserte kolonner1" = Table.ReorderColumns(#"Fjernede kolonner",{"Egendefinert.1", "Navn"}),
    #"Grupperte rader" = Table.Group(#"Omorganiserte kolonner1", {"Egendefinert.1"}, {{"Data", each _, type table [Egendefinert.1=text, Navn=nullable text]}}),
    #"Egendefinert lagt til2" = Table.AddColumn(#"Grupperte rader", "Komponenter", each Table.Column([Data],"Navn")),
    #"Uttrukne verdier" = Table.TransformColumns(#"Egendefinert lagt til2", {"Komponenter", each Text.Combine(List.Transform(_, Text.From), ";"), type text}),
    #"Aggregert Data" = Table.AggregateTableColumn(#"Uttrukne verdier", "Data", {{"Navn", List.Count, "Antall Navn"}}),
    #"Sorterte rader" = Table.Sort(#"Aggregert Data",{{"Egendefinert.1", Order.Ascending}}),
```

```
    #"Omorganiserte kolonner2" = Table.ReorderColumns(#"Sorterte
rader",{"Egendefinert.1", "Komponenter", "Antall Navn"})
in
    #"Omorganiserte kolonner2"
```

Appendix E EBL to RDS-PS mapping

1001	<Plant1.PS1.HP1>	0
1002	<Plant2.PS1.HP1>	0
1003	<Plant3.PS1.HP1>	0
1004	<Plant4.PS1.HP1>	0
1005	<Plant5.PS1.HP1>	0
1006	<Plant6.PS1.HP1>	0
1007	<Plant7.PS1.HP1>	0
1008	<Plant8.PS1.HP1>	0
1009	<Plant9.PS1.HP1>	0
1010	<Plant10.PS1.HP1>	0
1011	<Plant11.PS1.HP1>	0
1012	<Plant12.PS1.HP1>	0
1013	<Plant13.PS1.HP1>	0
1014	<Plant14.PS1.HP1>	0
1015	<Plant15.PS1.HP1>	0
1016	<Plant16.PS1.HP1>	0
1017	<Plant17.PS1.HP1>	0
1018	<Plant18.PS1.HP1>	0
1019	<Plant19.PS1.HP1>	0
1020	<Plant20.PS1.HP1>	0
1021	<Plant21.PS1.HP1>	0
311.!	X	1
311.!.079	X	1
311.!.079.010	X	1
311.!.079.011	X	1
311.!.079.012	X	1
311.!.079.030	X	1
311.!.079.031	X	1
311.!.079.032	X	1
311.!.301	X	1
311.!.302	X	1
311.!.303	X	1
311.!.304	X	1
311.!.305	X	1
311.!.306	X	1
311.!.307	X	1
311.!.308	X	1
311.!.309	X	1
311.!.310	X	1
311.!.311	X	1
311.!.312	X	1
311.!.313	X	1
311.!.314	X	1
311.!.315	X	1
311.!.316	X	1
312.!	X	1

312.!.079	X	1
312.!.079.010	X	1
312.!.079.011	X	1
312.!.079.012	X	1
312.!.079.050	X	1
314.!	X	1
314.!.210	X	1
314.!.220	X	1
314.!.230	X	1
314.!.240	X	1
314.!.250	X	1
314.!.260	X	1
314.!.270	X	1
314.!.270.100	X	1
314.!.300	X	1
314.!.310	X	1
314.!.410	X	1
314.!.510	X	1
314.!.610	X	1
314.!.701	X	1
315.!	C!	1
315.!.079	X	1
315.!.079.010	X	1
315.!.210	X	1
315.!.213	X	1
315.!.220	X	1
315.!.230	X	1
315.!.240	X	1
315.!.300	X	1
315.!.310	X	1
315.!.410	X	1
315.!.510	C!.JB5.HQB1	1
315.!.610	X	1
315.!.900	X	1
316.!	X	1
321.!	X	1
321.!.079	X	1
321.!.079.010	X	1
321.!.200	X	1
321.!.210	X	1
321.!.300	X	1
321.!.301	X	1
321.!.302	X	1
321.!.303	X	1
321.!.304	X	1
321.!.305	X	1
321.!.400	X	1
321.!.600	X	1
321.!.613	X	1

321.!.701	X	1
321.!.800	X	1
322.!	X	1
323.!	C!.JB5	1
323.!.200	X	1
323.!.301	X	1
323.!.302	X	1
323.!.400	X	1
323.!.500	X	1
323.!.510	X	1
323.!.520	X	1
323.!.530	X	1
323.!.540	X	1
323.!.550	X	1
323.!.710	X	1
323.!.800	X	1
324.!	X	1
324.!.079	X	1
324.!.079.010	X	1
327.!	X	1
327.!.110	X	1
401.!	A!.PG1	1
411.!	A!.RB1	1
411.!.110	A!.RB1.CNC1	1
411.!.110.300	X	1
411.!.120	X	1
411.!.210	A!.RB1.MLD1	1
411.!.220	X	1
411.!.230	X	1
411.!.240	X	1
411.!.300	X	1
411.!.310	A!.KA1.QNA1	1
411.!.316	A!.KA1.QPC1	1
411.!.320	X	1
411.!.410	A!.JB1	1
411.!.410.110	X	1
411.!.410.120	X	1
411.!.420	X	1
411.!.630	X	1
411.!.640	X	1
411.!.645	X	1
411.!.710	X	1
411.!.720	X	1
411.!.730	X	1
414.!	A!.KA1	1
414.!.200		1
414.!.300	A!.KA1	1
414.!.500	A!.KA1	1
415.!	X	1

415.!.200	A!.KA1.QMA1	1
415.!.500	X	1
415.!.600	X	1
417.!	X	1
421.!	A!.RA1	1
421.!.100	A!.RA1.RK1	1
421.!.200	A!.RA1.RJ1	1
421.!.250	X	1
421.!.260	A!.RA1.RJ1.XDB1	1
421.!.410	A!.RA1.RLC1	1
421.!.420	X	1
421.!.600	A!.RA1.HE1	1
421.!.700	X	1
421.!.710	A!.JF1.KJ1.UPA1	1
421.!.720	A!.JF1.KJ1.UPA2	1
421.!.900	X	1
424.!	A!.RA1.LD1	1
424.!.210	A!.RA1.LD1.TAA1	1
461.!	A!.JE1.KF1	1
461.!.200	X	1
462.!	X	1
483.!	X	1
485.!	D1!	1
486.!	C1!	1
487.!	H2	1
500.!	X	1
500.!.060	X	1
500.!.060.140	X	1
500.!.061	X	1
500.!.062	X	1
500.!.071	X	1
514.!	X	1
514.!.301	X	1
514.!.401	X	1
514.!.501	X	1
514.!.601	X	1
514.!.701	X	1
514.!.702	X	1
516.!	B132.JK!	1
516.!.301	A!.JE1.QAB	1
516.!.401	A!.JE1.QBB	1
516.!.402	A!.JE1.QCA	1
516.!.403	X	1
516.!.601	X	1
516.!.701	X	1
516.!.702	X	1
518.!	X	1
518.!.701	X	1
521.!	B22.JK!	1

521.!.301	A!.JE1.QAB	1
521.!.302	A!.JE1.QAB	1
521.!.401	A!.JE1.QCA	1
521.!.501	X	1
521.!.502	X	1
521.!.601	X	1
521.!.602	X	1
521.!.603	X	1
521.!.701	X	1
521.!.760	X	1
522.!	B11.JK!	1
522.!.100	X	1
522.!.301	A!.JE1.QAB	1
522.!.302	X	1
522.!.401	A!.JE1.QCA	1
522.!.402	X	1
522.!.501	X	1
522.!.502	X	1
522.!.503	X	1
522.!.601	X	1
522.!.602	X	1
522.!.603	X	1
522.!.701	X	1
522.!.760	X	1
522.!.781	X	1
523.!	B6.JK!	1
523.!.071.040	X	1
523.!.301	A!.JE1.QAB	1
523.!.401	A!.JE1.QCA	1
523.!.501	X	1
523.!.502	X	1
523.!.503	X	1
523.!.601	X	1
523.!.602	X	1
523.!.603	X	1
523.!.604	X	1
523.!.605	X	1
523.!.701	X	1
524.!	X	1
524.!.501	X	1
524.!.502	X	1
524.!.601	X	1
524.!.701	X	1
525.!	X	1
525.!.301	X	1
525.!.302	X	1
525.!.501	X	1
525.!.502	X	1
525.!.601	X	1

525.!.602	X	1
525.!.603	X	1
551.!	D1.JE!	1
551.!.301	D1.JE!.QAB	1
551.!.302	D1.JE!.QAB	1
551.!.303	X	1
551.!.304	X	1
551.!.305	X	1
551.!.306	X	1
551.!.307	X	1
551.!.308	X	1
551.!.309	X	1
552.!	D1.HD1.KF!	1
555.!	D2.JK!	1
556.!	D2.HD1	1
616.!	X	1
622.!	X	1
625.!	X	1
634.!	X	1
636.!	B123.HD!	1
641.!	B15.HD!	1
642.!	B11.HD!	1
643.!	B6.HD!	1
644.!	B3.HD!	1
645.!	B069.HD!	1
710.!	X	1
750.!	X	1
761.!	X	1
762.!	X	1
922.!	F1.PA1	1
923.!	D1.HF1	1
930.!	F1	1

Appendix F Python conversion script source code

The source code for the conversion script is contained within this appendix. Table 3 contains the “Tag” class, Table 4 contains the “RDS” class and Table 5 contains the EBL conversion function.

Table 3: Tag class source code

```
import re

class Tag:
    """Class to store RDS tag information and methods.

    Keyword arguments:
    tag_class -- RDS class
    serial_number -- RDS serial number
    name -- description
    level -- level in structure
    """
    def __init__(self, tag_class, serial_number, name, level):
        self.name = name
        self.tag_class = tag_class
        self.serial_number = serial_number
        self.tag = tag_class+str(serial_number) if serial_number != False
else tag_class
        self.level = level
        self.sources = {}
        self.childs = []

    def __eq__(self, other):
        if isinstance(other, Tag):
            return self.tag == other.tag
        return False

    def print(self):
        """Recursively print sub-tags to console."""
        for child in self.childs:
            jump = ""
            for i in range(0,child.level):
                jump += "\t"
            print(jump+child.tag+" - "+str(child.name)+" - From:
"+str(child.sources))
            child.print()

    def document_conversion(self, tag_rows, preceding):
        """Append RDS tag and source to existing list.
```

```
        Keyword arguments:
        tag_rows -- existing list to append rows to
        preceding -- tag above in the structure
        """
        if preceding != False:
            full_tag = preceding+"."+self.tag
        else:
            full_tag = self.tag
        tag_row = [full_tag, self.name, self.sources]
        tag_rows.append(tag_row)
        for i in range(0,len(self.childs)):
            self.childs[i].document_conversion(tag_rows, full_tag)

    def add_source(self, source_tag, source_system):
        """Add an converted source to tag."""
        if source_system in self.sources.keys():
            self.sources[source_system].append(source_tag)
        else:
            self.sources[source_system] = [source_tag]

    def find_child(self, tag):
        """Search for tag in sub-tags, return if found or False if not."""
        found = False
        for element in self.childs:
            if element.tag == tag:
                found = element
                return found
        return found

    def _find_next_serialnumber(self, tag):
        """Find next available serial number for tag, returns number."""
        used_numbers = [self.childs[i].serial_number for i, x in
enumerate(self.childs) if x.tag_class == tag]
        if not used_numbers:
            next_number = 1
        else:
            next_number = max(used_numbers)+1
        return next_number

    def _add_child_without_serialnumber(self,tag_class, name, level):
        """Add tag without a specified serial number, returns the created
tag.

        Keyword arguments:
        tag_class -- RDS class of the tag
        name -- description of the tag
```

```
        level -- level of the tag
        """
        serialnumber = self._find_next_serialnumber(tag_class)
        new_child = Tag(tag_class, serialnumber, name, level)
        self.childs.append(new_child)
        return new_child

    def _add_child_with_serialnumber(self,tag_class, serialnumber, name,
level):
        """Add tag with specified serial number, returns the created tag.

        Keyword arguments:
        tag_class -- RDS class of the tag
        serialnumber -- serial number
        name -- description of the tag
        level -- level of the tag
        """
        new_child = Tag(tag_class, serialnumber, name, level)
        self.childs.append(new_child)
        return new_child

    def add_child(self,tag,name,level):
        """Add tag, returns the created tag.

        Keyword arguments:
        tag -- RDS class of the tag
        name -- description of the tag
        level -- level of the tag
        """
        if "<" in tag:
            new_child = self._add_child_with_serialnumber(tag, False, name,
level)
            return new_child
        else:
            check_serialnumber = re.split(r"(\d+)",tag)
            check_serialnumber = list(filter(None, check_serialnumber))
            new_child = False
            if len(check_serialnumber)==1:
                new_child = self._add_child_without_serialnumber(tag, name,
level)
            elif len(check_serialnumber)==2:
                new_child =
self._add_child_with_serialnumber(check_serialnumber[0],
int(check_serialnumber[1]), name, level)
            else:
                #error with tag
                pass
```

```
            return new_child
```

Table 4: RDS class source code

```
import csv
from pandas import DataFrame
from tag import Tag

class RDS:
    """Class to control the conversion process."""
    def __init__(self):
        self.tags = Tag("Root","RDS",False,0)

    def print(self):
        """Print the current structure to the terminal."""
        print("Structure:")
        self.tags.print()

    def print_plant(self, top_node):
        """Print the structure for the plant to the terminal.

        Keyword arguments:
        top_node -- plant identifier
        """
        plant = self.tags.find_child(top_node)
        if plant != False:
            print(plant.tag+" - "+str(plant.name))
            plant.print()
        else:
            print("Not found")

    def save_conversion(self, filename, top_node = False):
        """Save and document the convertet structure as XLSX.

        Keyword arguments:
        filename -- name of the new file
        top_node -- optional plant identifier (default False)
        """
        top_child = False
        tag_rows = []
        if top_node == False:
            top_child = self.tags
            for plant in top_child.childs:
                tag_row = [plant.tag, plant.name, plant.sources]
                tag_rows.append(tag_row)
        else:
            plant = self.tags.find_child(top_node)
```

```
            if plant == False:
                top_child = []
                print("Plant identifier not found")
                return
            else:
                top_child = plant
                tag_row = [plant.tag, plant.name, plant.sources]
                tag_rows.append(tag_row)
        for i in range(0,len(top_child.childs)):
            top_child.childs[i].document_conversion(tag_rows, False if
top_node == False else top_child.tag)
        data = DataFrame(tag_rows, columns=['Tag','Name','Source'])
        data.to_excel(filename+".xlsx", index=False)

    def _load_csv(self, filepath, delimiter, limit = 100000):
        """Load rows from CSV and return as list.

        Keyword arguments:
        filepath -- file to load
        delimiter -- column delimiter
        limit -- optional maximum number of rows to read (default 100000)
        """
        data = []
        with open(filepath, newline='') as csvfile:
            reader = csv.reader(csvfile, delimiter=delimiter)
            counter = 0
            for row in reader:
                data.append(row)
                counter += 1
                if counter > limit:
                    print("Lastet ",len(data)," rader.")
                    return data
        print("Lastet ",len(data)," rader.")
        return data

    def _split_RDS(self, tag):
        """Seperate classes in RDS string to list.

        Keyword arguments:
        tag -- RDS tag
        """
        top_separation = tag.index(">")
        top_node = tag[:top_separation+1]
        classes = tag[top_separation+1:].split(".")
        if len(classes[0]) == 0:
            structure = [top_node]
        else:
```

```
            structure = [top_node]+classes
        return structure

    def load_RDS(self, filepath, delimiter):
        """Load RDS structure from CSV, with tags in first column and names
in second.

        Keyword arguments:
        filepath -- file to load
        delimiter -- column delimiter
        """
        rows = self._load_csv(filepath, delimiter)
        for i in range(0,len(rows)):
            tag = rows[i][0]
            structure = self._split_RDS(tag)
            self.add_tag(self.tags,structure,rows[i][1], 0)

    def add_tag(self, tag_node, tag, name, level = 0, source_tag = False,
source = False):
        """Add RDS tag to structure, including implicit tags.

        Keyword arguments:
        tag_node -- parent tag
        tag -- RDS tag
        name -- description
        level -- level counter (default 0)
        source_tag -- original tag (default False)
        source -- original tag standard (default False)
        """
        structure = tag
        if len(structure) == 1:
            child = tag_node.find_child(structure[0])
            if child == False:
                new_child = tag_node.add_child(structure[0], name, level)
                if source != False:
                    new_child.add_source(source_tag, source)
            else:
                #Updating existing tag
                if child.name == False:
                    child.name = name
                if source != False:
                    child.add_source(source_tag, source)
        else:
            remaining_structure = structure[1:]
            child = tag_node.find_child(structure[0])
            if child == False:
```

```
                new = tag_node.add_child(structure[0], name=False,
level=level)
                self.add_tag(new, remaining_structure, name, level+1,
source_tag, source)
            else:
                self.add_tag(child, remaining_structure, name, level+1,
source_tag, source)

    def convert_structure(self, datapath, mappingpath, delimiter,
converter):
        """Convert existing tags to RDS and add to structure.

        Keyword arguments:
        datapath -- path to CSV-file containing tags
        mappingpath -- path to CSV-file containing the mapping
        delimiter -- delimiter use with CSV-files
        converter -- function to convert the tags
        """
        rows_to_convert = self._load_csv(filepath=datapath,
delimiter=delimiter)
        mapping = self._load_csv(filepath=mappingpath, delimiter=delimiter)
        ebl_rows, source, source_rows = converter(rows_to_convert, mapping)
        print("Converted ",len(ebl_rows)," of ",len(rows_to_convert),"
tags")
        for i in range(0,len(ebl_rows)):
            tag = ebl_rows[i][0]
            structure = self._split_RDS(tag)
            self.add_tag(self.tags,structure,ebl_rows[i][1], 0,
source_rows[i], source)
```

Table 5: EBL conversion function source code

```
import re

def convert_EBL(original_tag, mapping):
    """Method for convertin EBL to RDS, returns list with RDS tags, source
standard and list of original tags.

    Keyword arguments:
    original_tag -- list of EBL information
    mapping -- list of mappings to RDS
    """
    source = "EBL"
    source_rows = []
    ebl_rows = []
```

```
    tags = [i[0] for i in original_tag]
    names = [i[1] for i in original_tag]
    top_nodes_tags = [i[0] for i in (t for t in mapping if int(t[4])==0)]
    top_nodes_ebl = [i[3] for i in (t for t in mapping if int(t[4])==0)]
    lower_nodes_tags = [i[0] for i in (t for t in mapping if int(t[4])==1)]
    lower_nodes_ebl = [i[3] for i in (t for t in mapping if int(t[4])==1)]
    for i in range (0, len(tags)):
        top = ""
        lower = ""
        try:
            top_ebl = tags[i][:4]
            lower_ebl = tags[i][5:]
            top = top_nodes_ebl[top_nodes_tags.index(top_ebl)]
            if len(lower_ebl) == 0:
                ebl_rows.append([top,names[i]])
                source_rows.append(tags[i])
            else:
                try:
                    substructure = lower_ebl.split(".")
                    if len(substructure) > 1:
                        serial_number = int(substructure[1])
                        substructure[1] = "!"
                        ebl_to_convert = ".".join(substructure)
                        lower =
lower_nodes_ebl[lower_nodes_tags.index(ebl_to_convert)].replace('!',
str(serial_number))
                        if len(lower) == 0:
                            #no conversion specified
                            pass
                        elif lower == "X":
                            #not relevant for RDS
                            pass
                        else:
                            ebl_tag = top+lower
                            ebl_rows.append([ebl_tag,names[i]])
                            source_rows.append(tags[i])
                except(ValueError):
                    #Class not found
                    pass
        except(ValueError):
            #Plant not found
            pass
    return ebl_rows, source, source_rows
```

Appendix G Python test scripts

This appendix contains the code used for testing the classes and functions in the Python script. Table 6 contains the code for testing the “Tag” class, Table 7 for the “RDS” class and Table 8 for the EBL conversion function.

Table 6: Code for testing the Tag class

```
from tag import Tag
import unittest

class TestTag(unittest.TestCase):
    def test_add_child_without_number(self):
        """
        Check that tag without number is created with the corret tag.
        """
        parent = Tag("<test1>",False,"Testplant",0)
        target = Tag("A1",False,"Testclass",1)
        test_tag = parent.add_child("A","Testclass",1)
        self.assertEqual(target,test_tag)

    def test_add_child_with_number(self):
        """
        Check that tag without number is created with the corret tag.
        """
        parent = Tag("<test1>",False,"Testplant",0)
        target = Tag("A2",False,"Testclass",1)
        test_tag = parent.add_child("A2","Testclass",1)
        self.assertEqual(target,test_tag)

    def test_find_next_serialnumber_no_existing(self):
        """
        Check that the next number is 1 when there i no existing class.
        """
        parent = Tag("<test1>",False,"Testplant",0)
        parent.add_child("A2","Testclass",1)
        target_number = 1
        test_number = parent._find_next_serialnumber("B")
        self.assertEqual(target_number,test_number)

    def test_find_next_serialnumber_existing_ordered(self):
        """
        Check that the next number is the next available.
        """
        parent = Tag("<test1>",False,"Testplant",0)
        parent.add_child("A1","Testclass1",1)
        parent.add_child("A2","Testclass2",1)
        target_number = 3
```

```
        test_number = parent._find_next_serialnumber("A")
        self.assertEqual(target_number,test_number)

    def test_find_next_serialnumber_exisiting_unordered(self):
        """
        Check that the next number is the next available.
        """
        parent = Tag("<test1>",False,"Testplant",0)
        parent.add_child("A1","Testclass1",1)
        parent.add_child("A3","Testclass2",1)
        target_number = 4
        test_number = parent._find_next_serialnumber("A")
        self.assertEqual(target_number,test_number)

    def test_find_child_existing(self):
        """
        Check that it finds the tag.
        """
        parent = Tag("<test1>",False,"Testplant",0)
        parent.add_child("A1","Testclass1",1)
        parent.add_child("A2","Testclass2",1)
        target_child = Tag("A2",False,"Testclass2",1)
        test_child = parent.find_child("A2")
        self.assertEqual(target_child,test_child)

    def test_find_child_nonexisting(self):
        """
        Check that it returns false.
        """
        parent = Tag("<test1>",False,"Testplant",0)
        parent.add_child("A1","Testclass1",1)
        parent.add_child("A2","Testclass2",1)
        target_child = False
        test_child = parent.find_child("A3")
        self.assertEqual(target_child,test_child)

    def test_add_source_no_source(self):
        """
        Check that the new source is added.
        """
        target = {"testsource": ["testtag"]}
        tag = Tag("<test1>",False,"Testplant",0)
        tag.add_source("testtag","testsource")
        test = tag.sources
        self.assertDictEqual(target,test)

    def test_add_source_additional_source(self):
```

```
        """
        Check that the new source is added.
        """
        target = {"testsource": ["testtag","testtag2"]}
        tag = Tag("<test1>",False,"Testplant",0)
        tag.add_source("testtag","testsource")
        tag.add_source("testtag2","testsource")
        test = tag.sources
        self.assertDictEqual(target,test)

    def test_add_source_new_source(self):
        """
        Check that the new source type is added.
        """
        target = {"testsource1":
["testtag","testtag2"],"testsource2":["testtag3"]}
        tag = Tag("<test1>",False,"Testplant",0)
        tag.add_source("testtag","testsource1")
        tag.add_source("testtag2","testsource1")
        tag.add_source("testtag3","testsource2")
        test = tag.sources
        self.assertDictEqual(target,test)

    def test_documentation(self):
        """
        Check that the documentation is formatted correctly.
        """
        target =
[["<test1>","testplant",{"test":["1001"]}],["<test1>.A1","testunit",{"test":
["1001.400"]}]]
        test = []
        test_struct = Tag("Root","RDS",False,0)
        plant = test_struct.add_child("<test1>","testplant",1)
        plant.add_source("1001","test")
        unit = plant.add_child("A1","testunit",2)
        unit.add_source("1001.400","test")
        plant.document_conversion(test, False)
        self.assertListEqual(target,test)

if __name__ == '__main__':
    unittest.main()
```

Table 7: Code for testing the RDS class

```
import unittest
from RDS import RDS

class TestRDS(unittest.TestCase):
    def test_split_RDS_only_topnode(self):
        """
        Check that topnode is returned.
        """
        rds = RDS()
        target = ["<Plant1>"]
        test = "<Plant1>"
        result = rds._split_RDS(test)
        self.assertListEqual(target,result)

    def test_split_RDS_topnode_multiple_levels(self):
        """
        Check that topnode and rest is returned.
        """
        rds = RDS()
        target = ["<Plant1>","A1","AB1"]
        test = "<Plant1>A1.AB1"
        result = rds._split_RDS(test)
        self.assertListEqual(target,result)

    def test_add_tag_one_level(self):
        """
        Test that the plant is added.
        """
        target = "<plant1>"
        test = RDS()
        test.add_tag(test.tags, ["<plant1>"], "testplant")
        result = test.tags.childs[0].tag
        self.assertEqual(target,result)

    def test_add_tag_second_level(self):
        """
        Test that tag is added to plant.
        """
        target = "<plant1>.A1"
        test = RDS()
        test.add_tag(test.tags, ["<plant1>"], "testplant")
        test.add_tag(test.tags, ["<plant1>","A1"], "testunit")
        result =
str(test.tags.childs[0].tag)+"."+str(test.tags.childs[0].childs[0].tag)
        self.assertEqual(target,result)
```

```
    def test_add_tag_implicit(self):
        """
        Test that the tag is added, inlcuding implicit tags.
        """
        target = "<plant1>.A1.AB1"
        test = RDS()
        test.add_tag(test.tags, ["<plant1>","A1","AB1"], "test")
        result =
str(test.tags.childs[0].tag)+"."+str(test.tags.childs[0].childs[0].tag)+"."+
str(test.tags.childs[0].childs[0].childs[0].tag)
        self.assertEqual(target,result)

    def test_add_tag_new(self):
        """
        Test that tag is addede to existing structure.
        """
        target = "<plant1>.B1.AB1"
        test = RDS()
        test.add_tag(test.tags, ["<plant1>","A1","AB1"], "test")
        test.add_tag(test.tags, ["<plant1>","B1","AB1"], "test2")
        result =
str(test.tags.childs[0].tag)+"."+str(test.tags.childs[0].childs[1].tag)+"."+
str(test.tags.childs[0].childs[1].childs[0].tag)
        self.assertEqual(target,result)

    def test_add_tag_copy(self):
        """
        Test that tag existing tag is updated.
        """
        target = "test2"
        test = RDS()
        test.add_tag(test.tags, ["<plant1>","A1","AB1"], "test")
        test.add_tag(test.tags, ["<plant1>","A1"], "test2")
        result = str(test.tags.childs[0].childs[0].name)
        self.assertEqual(target,result)

    def test_add_tag_no_serial(self):
        """
        Test that tag added with serial number.
        """
        target = "A1"
        test = RDS()
        test.add_tag(test.tags, ["<plant1>","A"], "test")
        result = str(test.tags.childs[0].childs[0].tag)
        self.assertEqual(target,result)

if __name__ == '__main__':
```

```
    unittest.main()
```

Table 8: Code for testing the EBL conversion function

```
from convert import convert_EBL
import unittest


class TestTag(unittest.TestCase):
    def setUp(self):
        self.testmapping =
[["1001","","","<plant1.PS1.HP1>","0"],["1002","","","<plant2.PS1.HP1>","0"]
,["421.!","","","A!.RA1","1"],["487.!","","","H2","1"]]
    def test_convert_plant(self):
        """
        Check that plant is converted.
        """
        target = [['<plant1.PS1.HP1>', 'testplant']], 'EBL', ['1001']
        test_tag = [["1001","testplant"]]
        result = convert_EBL(test_tag,self.testmapping)
        self.assertEqual(target,result)
    def test_convert_tag(self):
        """
        Check that tag with static serial is converted correctly.
        """
        target = [['<plant1.PS1.HP1>H2', 'bailing']], 'EBL', ['1001.487.1']
        test_tag = [["1001.487.1","bailing"]]
        result = convert_EBL(test_tag,self.testmapping)
        self.assertEqual(target,result)

    def test_convert_tag_convert_number(self):
        """
        Check that tag is converted correctly with number translation.
        """
        target = [['<plant1.PS1.HP1>A1.RA1',
'generator1'],['<plant1.PS1.HP1>A3.RA1', 'generator3']], 'EBL',
['1001.421.1', "1001.421.3"]
        test_tag = [["1001.421.1","generator1"],["1001.421.3","generator3"]]
        result = convert_EBL(test_tag,self.testmapping)
        self.assertEqual(target,result)

    def test_convert_tag_multiple_plants(self):
        """
        Check that tag is converted for multiple plants.
        """
```

```
        target = [['<plant1.PS1.HP1>A1.RA1',
'generator1'],['<plant2.PS1.HP1>A1.RA1', 'generator1']], 'EBL',
['1001.421.1', "1002.421.1"]
        test_tag = [["1001.421.1","generator1"],["1002.421.1","generator1"]]
        result = convert_EBL(test_tag,self.testmapping)
        self.assertEqual(target,result)

    def test_convert_plant_not_in_mapping(self):
        """
        Check that tag is not added if there is no match in plant.
        """
        target = [],"EBL",[]
        test_tag = [["1003.421.1","generator1"]]
        result = convert_EBL(test_tag,self.testmapping)
        self.assertEqual(target,result)

    def test_convert_component_not_in_mapping(self):
        """
        Check that tag is not added if there is no match in component.
        """
        target = [],"EBL",[]
        test_tag = [["1001.422.1","turb1"]]
        result = convert_EBL(test_tag,self.testmapping)
        self.assertEqual(target,result)

if __name__ == '__main__':
    unittest.main()
```