Data Validation of Spreadsheet Using Schema

An Application and a Schema to Support Validation of Large Amounts of Data in a Spreadsheet

Jacob Kronqvist

Degree Thesis for Bachelor of Engineering

Degree Programme in Information Technology

Vaasa 2018

EXAMENSARBETE

Författare:	Jacob Kronqvist
Utbildning och ort:	El- och automationsteknik, Vasa
Inriktningsalternativ/Fördjupning:	Informationsteknik
Handledare:	Susanne Österholm

Titel: Datavalidering av kalkylark med hjälp av schema – en applikation och ett schema för validering av stora mängder data i ett kalkylark

---

Datum: 7.6.2018 Sidantal: 29

Abstrakt

Uppgiften var att utveckla ett verktyg som importerar och validerar data ur ett kalkylark. Verktygets namn är Beamex Data Loader. Bakgrunden var att det behövdes ett verktyg som tar data från ett kalkylark med många rader data och för in det i Beamex CMX. CMX är kalibreringshanteringsmjukvara. För att förenkla kundens arbete behövs en lösning för att ersätta att kunden matar in data manuellt. Uppgiften utfördes för Beamex Oy Ab.

Uppgiften utfördes genom att använda teknologierna .NET Framework och eXtensible Markup Language (XML) scheman. Komponenter av .NET som användes var Language INtegrated Query (LINQ) samt Object Linking and Embedding, DataBase (OLE DB). Med dessa komponenter konverteras data i ett kalkylark till XML, och valideras med schema. För att hantera kalkylark användes även Microsoft Excel.

Resultatet blev en applikation som importerar data ur ett kalkylark och ett XML schema som data kan valideras med. För att försäkra att data är på rätt ställe i kalkylarket, utvecklades även en kalkylarksmall.

Språk: engelska Nyckelord: datavalidering, .NET, XSD, XML, Excel

---

OPINNÄYTETYÖ

Tekijä:	Jacob Kronqvist
Koulutus ja paikkakunta:	Tietotekniikka, Vaasa
Ohjaaja:	Susanne Österholm

Nimike: Laskentataulukon datan validointi skeeman avulla – Sovellus ja skeema suuren datamäärän validointiin laskentataulukossa

Päivämäärä: 7.6.2018 Sivumäärä: 29

Tiivistelmä

Tehtävänä oli kehittää työkalu, joka tuo laskentataulukon ja validoi dataa laskentataulukosta. Työkalun nimi on Beamex Data Loader. Taustalla on asiakkaiden ilmaisema tarve työkaluun, joka tuo laskentataulukon, jossa on monta riviä dataa, ja lisää dataa Beamex CMX:ään. Beamex CMX on kalibrointijärjestelmä. Helpottaakseen asiakkaan työtä tarvitaan ratkaisu, joka korvaa sen, että asiakas syöttää tiedot käsin. Tehtävä on tehty Beamex Oy Ab:lle.

Tehtävä on suoritettu käyttäen .NET Framework- ja eXtensible Markup Language (XML) -skeemoja. .NET:in käytetyt komponentit ovat Language INtegrated Query (LINQ) ja Object Linking and Embedding, DataBase (OLE DB). Käyttäen näitä komponentteja laskentataulukko muunnetaan XML:ksi ja validoidaan käyttämällä XML skeemaa. Laskentataulukkojen hallinta tehtiin käyttämällä Microsoft Excel-ohjelmistoa.

Tulokseksi saatiin sovellus, joka tuo dataa taulukkolaskennasta sekä XML-skeema, jolla voidaan validoida. Lisäksi kehitettiin taulukkolaskentamalli, jolla varmistetaan, että data on taulukossa oikeilla paikoillaan.

Kieli: englanti Avainsanat: datavalidointi, .NET, XSD, XML, Excel

BACHELOR'S THESIS

Author:	Jacob Kronqvist
Degree Program:	Engineer (Bachelor of Engineering)
Specialization:	Information technology
Supervisor(s):	Susanne Österholm

Title: Data Validation of Spreadsheet Using Schema – An Application and a Schema to Support Validation of Large Amounts of Data in a Spreadsheet

---

Date: 7.6.2018 Number of pages: 29

Abstract

The assignment was to develop a tool that imports and validates data from a spreadsheet. The name of the tool is Beamex Data Loader. The background is that customers have expressed the need for a tool that imports data from a spreadsheet with many rows of data and inserts the data into Beamex CMX. CMX is calibration management software. To make the work for the customer easier a solution to entering data manually is needed. The assignment was done for Beamex Oy Ab.

The assignment was done using the technologies .NET Framework and eXtensible Markup Language (XML) schemas. Components of .NET that were used are Language INtegrated Query (LINQ) and Object Linking and Embedding, DataBase (OLE DB). With these components data in a spreadsheet is converted to XML and validated with a schema. Spreadsheet handling was done using Microsoft Excel.

Finally, an application was developed that takes a spreadsheet and an XML schema to validate the data with. To ensure the data is in the right position in the spreadsheet, a spreadsheet template was also developed.

Language: English Key words: Data validation, .NET, XSD, XML, Excel

---

Contents

ABBREVIATIONS & DESCRIPTIONS

Abbreviation	Description
API	Application Programming Interface
B3	Beamex Business Bridge
CLR	Common Language Runtime
CMX	Calibration management software
CWSI	Common Web Services Interface
DTD	Document Type Definition
HTML	HyperText Markup Language
ICS	Integrated Calibration Solution
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
LINQ	Language Integrated Query
LOGiCAL	Cloud based calibration management software
OLE DB	Object Linking and Embedding, DataBase
PM	Product Manager
PO	Product Owner
R&D	Research and Development
RELAX NG	REgular LAnguage for XML Next Generation
SOP	Standard Operating Procedure
SQL	Structured Query Language
UI	User Interface
URS	User Requirement Specification
W3C	World Wide Web Consortium
VB	Visual Basic
VBA	Visual Basic for Applications
XML	eXtensible Markup Language
XSD	XML Schema Definition

1 Introduction

In this chapter a brief introduction to the employer and assignment will be made.

1.1 Employer

Beamex Oy Ab is a company based in Pietarsaari, Ostrobothnia Finland. They are part of Sarlin Group Oy Ab. Their main products are calibrators and calibration software. Beamex has customers all over the world and subsidiaries in many countries.

They provide an Integrated Calibration Solution (ICS) which involves software, hardware and expertise. This solution allows for paperless calibration (Figure 1). Their customers include pharmaceutical, power generation and chemical processing companies. (Beamex Oy Ab, 2018)

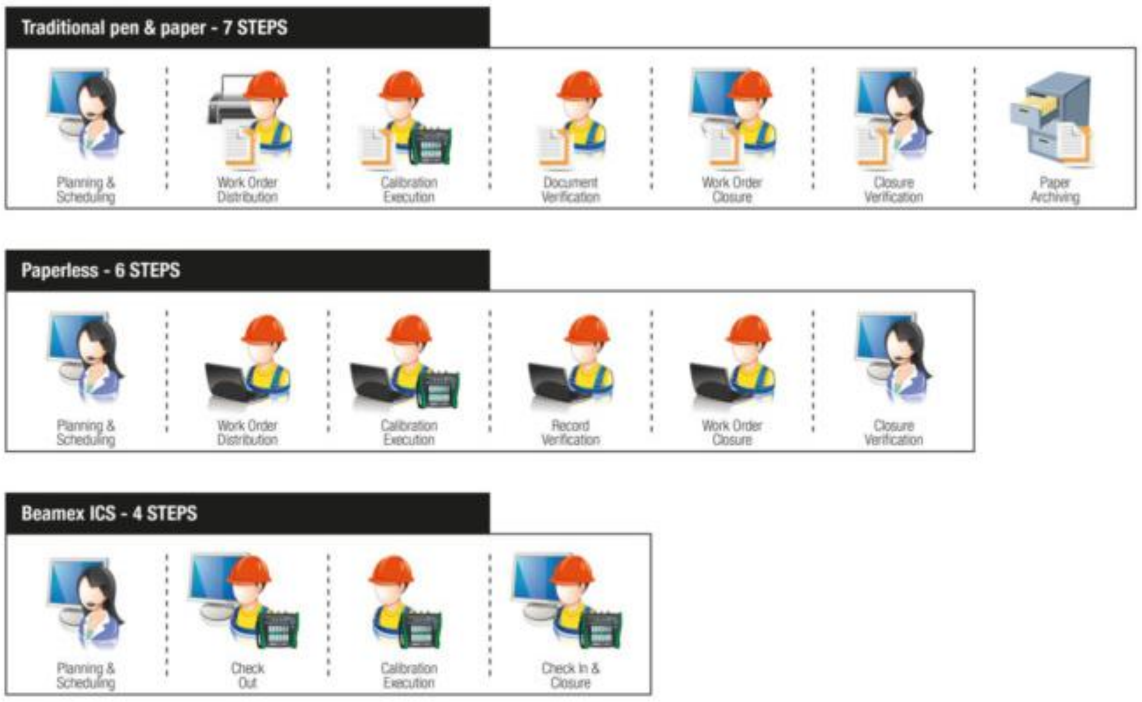


Figure 1. Comparison of calibration solutions (Beamex Oy Ab, 2018)

1.2 Assignment

The assignment was to create an application that validates data in a spreadsheet. The name of the application will be Beamex Data Loader. The application will be integrated with calibration software from the same company, CMX, where it will be useful for the customer.

The basic functionality is: the user selects a spreadsheet in the application. The application then makes sure that the data in the spreadsheet is correct and displays the errors to the user. This allows the user to correct the data in the spreadsheet. It also makes sure the data is correct when, for example, inserting it into the CMX database. If the data is correct, it is inserted into a database using CMX. It must support communication with CMX (Figure 2).

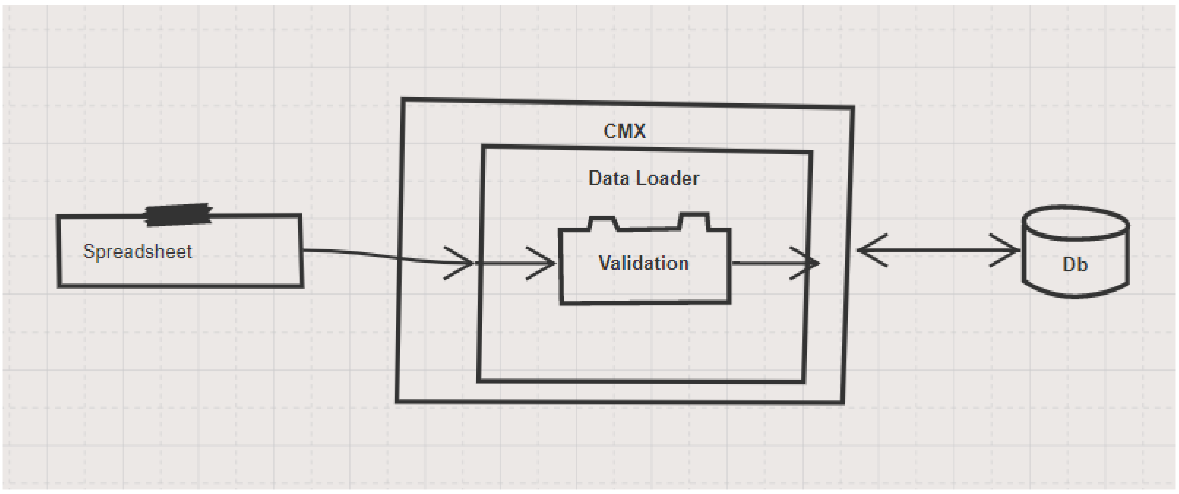


Figure 2. Illustration of the basic functionality of Data Loader

1.3 Background

As the customers' databases grow bigger, it is challenging to keep track of data in them. Some data is stored in spreadsheets with up to tens of thousands of rows. So, they have expressed the need for a tool to handle larger amounts of data. Every customer that has expressed this need uses spreadsheets for keeping track of some data already.

The previous solution is a working remedy for these issues, however, it was made for internal use only. This solution will be discussed in Chapter 2.1.1.

2 Approach

2.1 Prestudy

To really gauge the assignment some research prior to actual thesis work had to be made. Mostly to know if the subject was suitable for a thesis.

2.1.1 Previous solution

The previous solution is a solution for customers' demands on a tool that can handle large amounts of data. This tool will be replaced with the new Data Loader. It is created for some of Beamex's customers and professional services. More on professional services in chapter 2.5. Since the new solution will replace this tool, it is a good place to start to look for ideas. This tool, however, cannot be released because it only was made for internal use. There are also some other reasons why a new solution must be developed from scratch, more on this later in this chapter.

This tool works in many similar ways as the solution that is going to replace it. Its basic functionality is similar (Figure 3): a user selects a spreadsheet, the tool parses it and inserts the data into a database. It also uses a schema for validating the data in the spreadsheet, and the results of the validation are shown to the user in a list box. The spreadsheet is made using a template with all the supported calibration fields in the right order. The user can then add all the data they want to the correct fields. It also uses the same database as the new solution, however, through a different route.

The major flaw of this tool is the communication, which is directly with the database. This means that it cannot use the credentials of the calibration software without implementing a handler for it. If these credentials are not used, the audit trail will be surpassed, which is an important feature supported in almost every Beamex product. It is also dangerous to have direct communication with a database in the hands of an inexperienced user.

Implementing these features into the previous solution could be possible, however, it was decided that a new solution made for commercial use and according to Beamex's coding and design standards would be a better option.

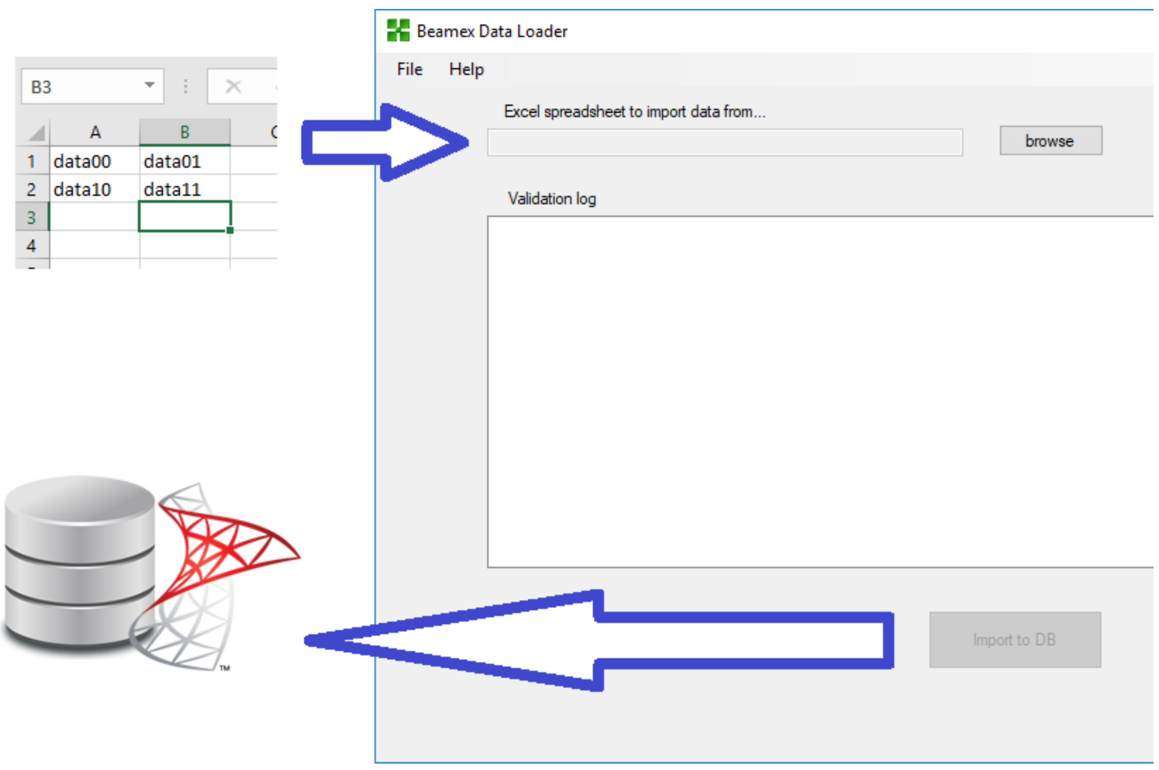


Figure 3. Illustration of the basic functionality of previous solution

2.1.2 Beamex Standard Operating Procedure (SOP)

Like any other software company SOPs have been specified at Beamex to keep the development effective and consistent. The SOPs applicable for the thesis were: coding standard, project management and process description. They specified what documents to write and how to avoid critical errors in development.

2.2 Limitations

To make sure all the entities affected by Data Loader had a say in its development, planning meetings were required. During most meetings the Product Manager (PM), the Product Owner (PO) of Data Loader and CMX, and the PO of LOGiCAL were present.

Data Loader it too big for just one developer, so, the project was split in two parts. The first part of the development process was the thesis, researching and developing a solution for the basic requirements. The second part is where incremental additions and improvements are added to the solution to eventually meet all the requirements.

2.2.1 User Requirement Specification (URS)

The URS is a document that specifies what is required and what should be in the solution. It is written by the PM. The document consists mostly of requirements specified by customers. Feedback from customers is fed in to the quality feedback database where the needs and issues they come up with are recorded. The requirements related to Data Loader have been gathered and specified in the URS. This document is a work in progress and will be changed during the development process of Data Loader. Some requirements may, for example, be scrapped due to not being feasible or due to a tighter schedule.

This document contains many requirements, both general and Data Loader specific. The general requirements could be applied to any software from any company. Here requirements like name, looks and general functionality are specified. In the case of Data Loader, the official product name should be Beamex Data Loader, it must look like a Beamex product and it should be intuitive to use. But also, some specifics like that it should be a stand-alone tool and not limited by any other software, for example the user is not required to have Microsoft Excel installed on their computer. It should also perform better than the previous solution.

The functionality requirements are: Data Loader must be compatible with CMX, must not communicate directly with a database but through CMX. It must not require other software to be installed or be installed itself to use it. But since it is going to communicate through CMX, CMX must be installed. It must have accessibility control, users who do not have permission to use it should not be able to.

Then more specific requirements are listed, of which the most relevant ones are:

> *“*
> *6.3.2. R3.2. Data loader MUST be able to validate the data for being technically correct before it can be imported to CMX.*
> *If there is a problem with some instrument in the data validation, it should still be possible to continue to the import phase (import data that was successfully validated).”*
> *6.6.1. R6.1. The import template file MUST be of a format that is commonly available*
> *For example, current data loader template is in Excel format. ”*

2.3 Technologies

2.3.1 .NET Framework

.NET is a software framework developed by Microsoft for Microsoft Windows, but some versions work on other platforms as well. One of the benefits of using this framework is its large, framework class library, which allows developers to use their own code with pre-existing code.

Other benefits derive from Common Language Runtime (CLR), which makes security, memory management and exception handling among other things easier. This is possible because CLR runs in a software environment using virtual machines, in comparison to hardware in other programming languages. It works with many programming languages, even allowing communication between them (Figure 4). Because of these features code written with .NET can be called managed code.

Traditionally .NET was developed proprietary, but has since then evolved more and more into being community developed. (Microsoft, 2018)

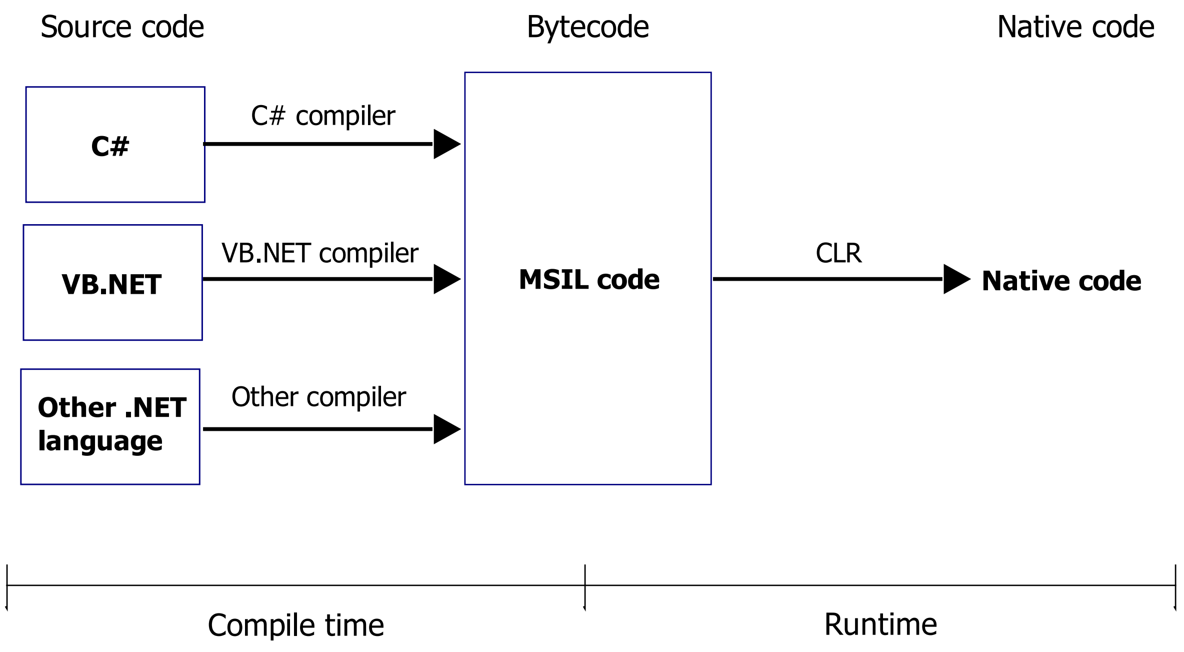


Figure 4. Illustration of CLR (Storset, 2007)

2.3.2 Microsoft Visual Studio

Microsoft Visual Studio is an Integrated Development Environment (IDE) which is used by .NET developers among others. The newest version, 2017, was used when developing Data Loader. The integrated debugger was used extensively in the development.

2.3.3 Language INtegrated Query (LINQ)

LINQ is a component of .NET and is used for data querying. The statements resemble Structured Query Language (SQL) statements, for example “SELECT * FROM”. Data used within an application is usually object oriented, but some sources of external data may not be, for example eXtensible Markup Language (XML) or relational databases. LINQ aims to make this data accessible. By using general purpose queries for all data sources, the process of accessing data is simplified. (Box & Hejlsberg, 2007) (Microsoft, 2018)

LINQ to XML is a provider for LINQ that can be used as an Application Programming Interface (API) for XML programming. It can covert XMLs to a collection of XElement objects. These objects can then be used to design an XML tree with multiple levels. This feature was used extensively in the development of Data Loader. LINQ to XML is also capable of reading and saving XMLs. (Champion, 2007)

2.3.4 Object Linking and Embedding, DataBase (OLE DB)

OLE DB is an API made by Microsoft for accessing data in a large variety of sources, including, for example spreadsheets. It also separates the data source from the application through abstractions to make the syntax as similar for different applications and different data sources as possible. It is part of the Microsoft data access components. Users of it can be divided into providers and consumers, and as the names may suggest providers give data to the consumers who, in turn, use the data. (Microsoft, 2018)

2.3.5 Microsoft Excel

Excel is a spreadsheet application developed by Microsoft. It works like any other spreadsheet application by using grids with cells. Some important functions for data analysis are included, but it also supports a wide range of add-ins. Programming is also supported using Visual Basic for Applications. Even though it has some features that allow it to be used as a database, for example Format as Table, this is not its intended use.

2.3.6 eXtensible Markup Language (XML)

XML is a markup language developed by the World Wide Web Consortium (W3C), it is both human and machine readable and an open standard. It is made to store data in comparison to other markup languages like HyperText Markup Language (HTML) which view it. It does not contain any predefined tags. It has also been the foundation for many other formats like OpenDocument. The design goals for XML are:

“

1. *XML shall be straightforwardly usable over the Internet.*
2. *XML shall support a wide variety of applications.*
3. *XML shall be compatible with SGML.*
4. *It shall be easy to write programs which process XML documents.*
5. *The number of optional features in XML is to be kept to the absolute minimum, ideally zero.*
6. *XML documents should be human-legible and reasonably clear.*
7. *The XML design should be prepared quickly.*
8. *The design of XML shall be formal and concise.*
9. *XML documents shall be easy to create.*
10. *Terseness in XML markup is of minimal importance.* *“*

(W3C, 2008)

To use XMLs you need a processor or parser, and there are many APIs developed for this purpose. XMLs are constructed using Unicode characters with angle brackets as tags (Code e.g. 1). One criticism is the complexity of the language because of its extensive use of angle brackets. (Atwood, 2008)

Code e.g. 1

```xml
<?xml version="1.0" encoding="UTF-8"?>
<item>
  <description>Computer</description>
  <price>499.99</price>
  <heading>Computer</heading>
  <body>The latest and greatest gaming PC!</body>
</item>
```

Error handling in XML is non- forgiving and the parser will report any error in comparison to HTML where the parser always tries its best to view the markup. For an XML to be valid it must contain legal characters, correct nesting and have a single root element etc. Fortunately, there are many validators either built in the parser or online embedded in a web page. The validity can also be checked using XML schemas to compare it to. (Walsh, 1998)

2.3.7 XML schemas

Schemas define constraints to the structure and content of an XML. It uses a similar syntax as XMLs: elements and attributes, data types, default and fixed values, for validation. The basic principle is: a user enters data into an XML, an application compares it to the rules of the schema to make sure it is valid or not. Using the XML in code e.g. 1 and the schema in code e.g. 2 we get the validation error:" '499.99' is not a valid value for 'integer'". There are multiple types of schemas, the most used are XML Schema Definition (XSD), Document Type Definition (DTD) and REgular LAnguage for XML Next Generation (RELAX NG). DTD is the most supported, but not as strong as XSD which has support for more functions. (van der Vlist, 2001) The W3C Schema (XSD) Validator online is a useful tool if you want to debug your XML and schema. (Briganti, 2018)

Code e.g. 2

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="item">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="description" type="xs:string"/>
      <xs:element name="price" type="xs:int"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

2.4 Related Beamex products

2.4.1 CMX

CMX is calibration software developed by Beamex. This software is the centerpiece of the ICS package. This software is used to handle everything from calibrators to security. It is mostly written in .NET and VB.NET. The instrument structure in CMX refers to all the fields that are available when calibrating, for example when using an electrical indicator, you can add calibration points in the unit voltage to a field, and a note describing where it was done to another field. CMX supports a lot of different fields ranging from text to numbers, and will support even more in the future.

2.4.2 LOGiCAL

Logical is also calibration software developed by Beamex. This software is cloud based. It is most likely going to replace CMX someday.

2.4.3 Business Bridge (B3)

Beamex Business Bridge is a helper application where customers can import data into CMX database with XMLs. The application is polling for files and is running in the background and will handle any XML in a specific location. It can do this remotely since it’s communicating directly with the database as opposed to CMX.

2.4.4 Common Web Services Interface (CWSI)

Communication protocol for various Beamex products, works over the internet. Instead of having to plug your calibrator to the computer with CMX you can install a client on another computer with a calibrator plugged into it and communicate with CMX over the internet. Current calibrators only work over USB and can only be connected to the internet through a computer. With this software you do not have to have physical access to the calibrator.

2.5 Professional Services

Professional services is a branch at Beamex that has direct contact with customers unlike Research & Development (R&D). They, together with the customer and sometimes the customer's data figure out the best solutions to fit the customer's needs. They are responsible for discovering some bugs and testing for R&D.

3 Development

3.1 Planning

This chapter will explain how planning was done for Data Loader. Planning usually involved finding solutions to a specific requirement, presenting the solutions in a meeting and deciding which solution was the best. The planning can be divided into five parts.

In the first part, research was done on ways to communicate with calibration software. The options were direct communication, using B3 or CWSI. It was decided that B3 would be the most fitting technology, because it supported remote communication and had support for most of the instrument structure, unlike CWSI. But also, because a module could be added to B3 to support communication with LOGiCAL later.

The second part involved research on solutions that validate the data in a spreadsheet. The spreadsheet used in B3 to create XMLs had data validation. It worked by using the export function in Excel. This idea was abandoned since the XML produced was not user friendly enough. Focusing on the structure of the XML was next.

In the third part of planning, a rough model of the instrument structure was modeled in an XML tree, to overcome the shortcomings of the last solution. The model had a greater tree depth to improve user friendliness compared to the XML produced by Excel. This was done using the schema from the previous solution of Data Loader. The solution was accepted, since the XML was a lot more user friendly than the one produced with Excel.

In the fourth part of planning it was decided, that due to a tighter schedule, Data Loader would be developed using direct communication with CMX as opposed to the previous decision, B3. It would also be a part of CMX as opposed to a stand-alone application.

The fifth part involved researching solutions that support spreadsheet parsing and exporting XMLs with multiple levels. One option here was to use the built-in components of .NET. So, a demo was presented, and the idea was accepted, since it would use the same technology as CMX and therefore guarantee compatibility. The requirements from the URS and affected parties were also met.

3.2 Excel export to XML function

Excel has a built-in function to export data in spreadsheet directly to an XML. In the first part of the development process, this function is studied. Could a schema and Excel be sufficient for the needs of our customers and for future products? It was determined that the flat structure of the XML generated by the function would not be sufficient (Figure 5). This means that the function will not be used to generate XML, instead an application will parse the spreadsheet directly. Excel will, however, be used to collect calibration data. Excel can after all handle multiple levels in a tree, but not export it to an XML. A lot of research and development was done on this function, even though it was finally dismissed.

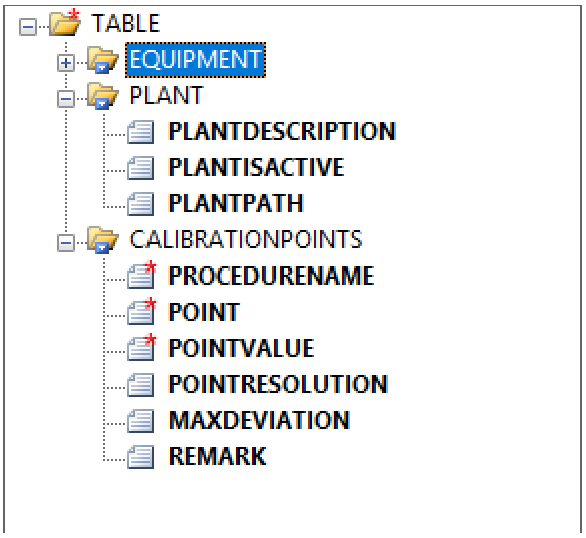


Figure 5. Illustration of the flat tree in Excel

3.2.1 User Interface (UI)

Customers can just create a raw XML and import it directly into CMX using B3, however, this is not user friendly. An instrument with all its related data can contain hundreds of fields. Therefore, Excel was adopted. Customers are using spreadsheets to keep track of some of their instrument data. Excel also has an export function, as mentioned earlier, which with the help of a schema creates an XML suited for importing to CMX. The UI in a spreadsheet is way better than editing XML raw.

3.2.2 B3 XML Creator

A spreadsheet for creating instruments in XMLs already existed for B3. B3 XML Creator is an Excel spreadsheet developed to make it easier to create valid XMLs for B3. Editing XMLs can be time consuming and is not very user friendly. Validating it according to a schema can also be tricky. This spreadsheet contains most of the instrument fields in CMX and an improved UI. So, this spreadsheet was used as a base when researching the Excel export to XML function. The task was to make this spreadsheet capable of creating many instruments in a single XML.

3.2.3 Multiple elements

The main issue with the B3 XML Creator spreadsheet and its schema was that it only supported one instrument per exported XML. An attempt to bind a field from the tree (Figure 5) to multiple data points in the sheet was done, but Excel does not support that. (Bohring & Auer, 2005) Linking the field to a list was tried, however this was not successful either. Finally, editing the schema was successful. All that had to be done was to implement a list of all instruments (Code e.g. 3). The instrument element contains all the fields required and the root element, instruments, lists all the entities of the instrument element. (Mowry, 2015)

Code e.g. 3

```
<xs:element name="INSTRUMENTS">
            <xs:complexType>
            <xs:sequence>
                        <xs:element ref="INSTRUMENT" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            </xs:complexType>
</xs:element>
```

3.2.4 Data Validation

To make it easier for the user to add valid data, validation in the spreadsheet was also considered. The existing data validation happened only in the process of exporting, with a schema. When handling multiple rows, this could be a problem if the user does not realize that the data entered is invalid. After entering many rows of data, redoing everything would be time consuming. However, the dialog that appears when exporting in Excel, is quite straight forward. So, finding and fixing errors using that is possible. A decision against adding validation in the spreadsheet was made because it would require editing of both the schema and the spreadsheet if any changes to the validation had to be made.

3.2.5 Abandoning the function

The instrument structure used by CMX requires multiple levels in a tree. The only workaround for having many levels, using the Excel export function, was to put different nodes that require levels directly below the top node and using identifiers to connect the data. So, the top node, instruments would contain the instrument structure like before, but also calibration points structure etc. This is not optimal due to having to add identifier fields to many elements in the instrument structure. This would not make the XML very user readable either, even though no one is going to edit the XML raw. Considering the incompatibility of the function, it was finally abandoned in favor of a solution that would natively support multiple levels in a tree in an XML.

3.3 Schema

It is recommended to create a valid XML to model the schema, when developing schemas. Especially if you do not know the layout that the XML should have well. This allows you to look at the structure instead of focusing on validation right away.

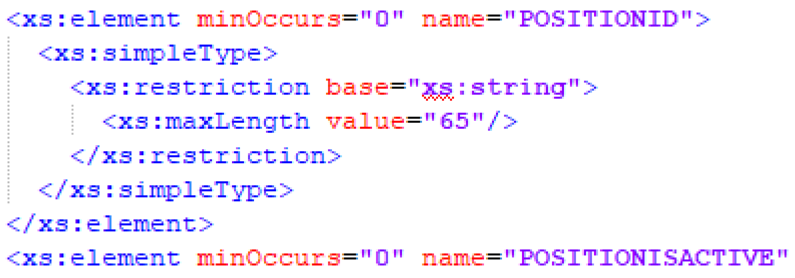
3.3.1 Schema for previous Data Loader

Because the schema made for the previous Data Loader already contains most of the validation required, it can be used as a base for the new schema. It also contains most of the instrument structure.

3.3.2 Adding levels

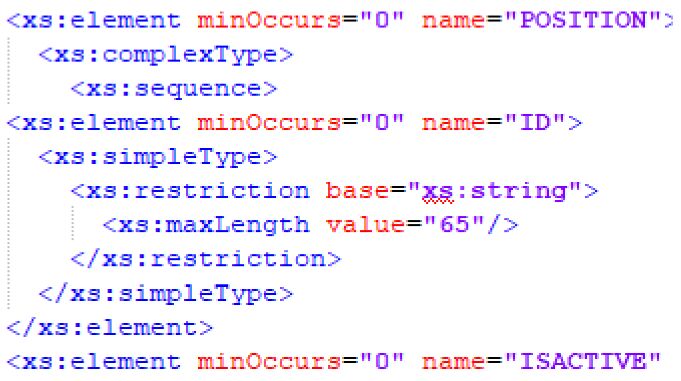
If the schema is user friendly, the XML will also be more user friendly. A flat structure on any kind of XML is not user friendly when dealing with many elements. Also, since the schema no longer is limited to the amount of levels, basic levels can be added to make it more user friendly. This is done by, for example creating a node from an element and moving all elements related to it below it. In code e.g. 4 you can see the code before the change and in code e.g. 5 you can see the code after the change.

Code e.g. 4



```
<xs:element minOccurs="0" name="POSITIONID">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:maxLength value="65"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element minOccurs="0" name="POSITIONISACTIVE"
```

Code e.g. 5

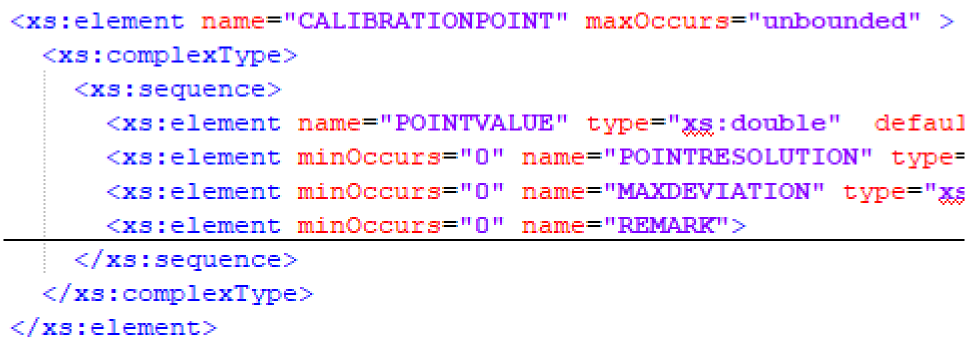


```
<xs:element minOccurs="0" name="POSITION">
  <xs:complexType>
    <xs:sequence>
<xs:element minOccurs="0" name="ID">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:maxLength value="65"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element minOccurs="0" name="ISACTIVE"
```

3.3.3 Making static elements dynamic

Some of the elements in the instrument structure were statically limited. For example, calibration points in a calibration, usually they are fewer than 18, but can basically be infinite. This was solved by making the maxOccurs attribute unbounded. (Code e.g. 6) Making the elements dynamic can also help in the future if the customer asks for a number of elements that is more than was previously available.

Code e.g. 6

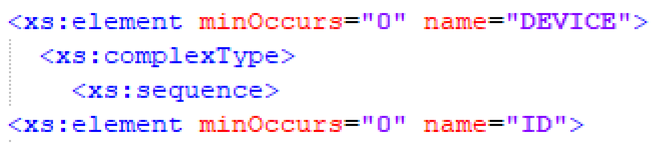


```
<xs:element name="CALIBRATIONPOINT" maxOccurs="unbounded" >
  <xs:complexType>
    <xs:sequence>
      <xs:element name="POINTVALUE" type="xs:double"  defaul
      <xs:element minOccurs="0" name="POINTRESOLUTION" type=
      <xs:element minOccurs="0" name="MAXDEVIATION" type="xs
      <xs:element minOccurs="0" name="REMARK">
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

3.3.4 Renaming elements

Because the schema has a more treelike structure we can have similar element names for different elements under different nodes. For example, the position node in code e.g. 5 and the device node in code e.g. 7 both have an element below them called ID. The XML definition states that you cannot have two elements with the same name under the same node in a valid schema. However, now that we have multiple levels, it does not affect us. This makes the code more readable. Especially when you can remove the prefix for every element name. For example, in code e.g. 4 you can see the element POSITIONID become ID in code e.g. 5.

Code e.g. 7



```
<xs:element minOccurs="0" name="DEVICE">
  <xs:complexType>
    <xs:sequence>
<xs:element minOccurs="0" name="ID">
```

3.3.5 Adding new elements

Since a new schema had to be developed, adding some new elements to the instrument structure for future support could also be done at the same time. New fields in the instrument structure are added on every release of CMX to support new functions.

3.4 Application

Many solutions for validating spreadsheets exist. However, if you want to validate using a schema, you must parse the data in the spreadsheet and then create an XML from the data. A schema does not support validation of anything else but an XML. In this case the XML does not just exist in the application but is also created in the application directory. This allows the possibility to validate the XML using, for example online validators.

3.4.1 Simple demo

To figure out whether it was possible to handle spreadsheets in a separate application, a demo of the functionality had to be found. .NET has a lot of different components that support handling of spreadsheets. A demo was found on the internet (Behera, 2017) and broken down.

3.4.2 Validation mode

To make the demo work, some changes had to be made to the code. It started by validating DTDs which was not applicable for Data Loader, so figuring out how to make it validate XSDs was necessary. It took a while to figure out how to make it validate using an XSD instead of the other schema formats, since the application seemed to default to DTD unless all variables necessary for XSD were defined (Code e.g. 8). By now a working demo application with demo data was developed.

Code e.g. 8

```
XmlReaderSettings xmlReaderSettings = new XmlReaderSettings{
ValidationType = ValidationType.Schema
};
xmlReaderSettings.Schemas.Add("urn:instruments-schema", sPath + "\\EquipmentImport.xsd");
```

3.4.3 Spreadsheet parsing

Spreadsheet parsing works by first accessing the spreadsheet data with the OLE DB interface. This works similarly to accessing a database. The command text selects all data from the spreadsheet. Using the OLE DB Data Adapter with the command text the data in the spreadsheet can be filled into a DataTable object. (Code e.g. 9) Now that we have a DataTable it can be used just like a spreadsheet, and commands to figure out what data exists in what cells are easily called. To get the correct structure for the XML, the XObject class with XElement and XAttribute was used. When all the required cells are parsed into the structure, it can be saved using the method Save. This creates an XML which can be validated using a schema.

Code e.g. 9

```
oleDbCommand.CommandText = "SELECT * FROM [" + sSheetName + "]";

DataTable dataTable = new DataTable {
TableName = sSheetName.Replace("$", string.Empty)
};

OleDbDataAdapter dataAdapter = new OleDbDataAdapter(oleDbCommand);
dataAdapter.Fill(dataTable);

dataSet.Tables.Add(dataTable);
```

One thing all the solutions researched had in common was that the spreadsheet first had to be converted into an XML to support validation with a schema. The demo used could convert a spreadsheet to XML, however it did not have any kind of structure, so it was just as useless as Excels export to XML function. This was solved by creating a tree for handling all the fields in the spreadsheet. (Code e.g. 10)

Code e.g. 10

```
new XElement(xNamespace + "CHECKLISTGROUP",
new XElement(xNamespace + "CHECKLISTGROUP",
new XElement(xNamespace + "NAME", dataTable.Rows[i][60]),
new XElement(xNamespace + "CHECKLISTITEM",
new XElement(xNamespace + "CHECKLISTITEM",
new XElement(xNamespace + "ID", dataTable.Rows[i][60]),
new XElement(xNamespace + "DESCRIPTION", dataTable.Rows[i][60])))))
```

3.4.4 Validation

The validation is done using XmlReader. XmlReader reads the XML row by row and calls upon an event handler with the results, if an error is found. (Code e.g. 11)

Code e.g. 11

```
xmlReaderSettings.ValidationEventHandler += new ValidationEventHandler(ValidationCallBack);

XmlReader xmlReader = XmlReader.Create(xmlNodeReader, xmlReaderSettings);

while (xmlReader.Read()) ;
```

3.4.5 Multiple instruments

When handling instrument data in spreadsheets you use multiple rows, so, supporting this in the application is paramount. This was solved by moving the parsing of the spreadsheet into a loop. Every element parsing cells had to get a dynamic row number too. (Code e.g. 12)

Code e.g. 12

```
var treeNode = new XElement(xNamespace + "INSTRUMENTS");
for (int i = 0; i < dataTable.Rows.Count; i++){
treeNode.Add(
new XElement(xNamespace + "INSTRUMENT",
new XAttribute("version", dataTable.Rows[i][0]),
```

3.4.6 Excel template

The application cannot accept any spreadsheet because the code is statically bound to columns in the spreadsheet, for example if you inserted a column as the first column in the spreadsheet, all the data would be moved one index to the right. This is not good software design and will be discussed in chapter 5.2.1. But for now, this is fixed by making a spreadsheet template. The template can be sent to customers to fill with data. Not all elements in the instrument structure can be static however. The number of calibration points can be anywhere from 0 to 200, and maybe more in the future. This is going to be a challenge to implement.

Since the instrument structure contains many elements, making a good UI is hard. Putting the different nodes in the instrument structure on different sheets in the spreadsheet would improve the UI. This would make sure the spreadsheet is not going to be hundreds of columns wide. The instrument structure has some patterns considering which elements will be used with a certain instrument type, for example it is unnecessary to have a field for pressure when using an electrical indicator, all the data related to pressure instruments could be moved to one sheet and all the data related to electrical instruments to another sheet. This can be used to make sheets for certain instruments that contain only the relevant fields, to further improve UI.

The template for the previous Data Loader will come to good use. Using this template and writing code based on its layout has a couple of benefits: Customers would not have to update their templates, and if they already have data in them it is going to work out of the box. Customers are also used to the template and keeping it would not require any additional training. Some customers may even have software statically bound to the cells in the spreadsheet, and using the same template would make sure that they do not have to update their software. But changes will have to be made to the template at some point due to updates in the instrument structure in CMX.

3.5 Choice of technology

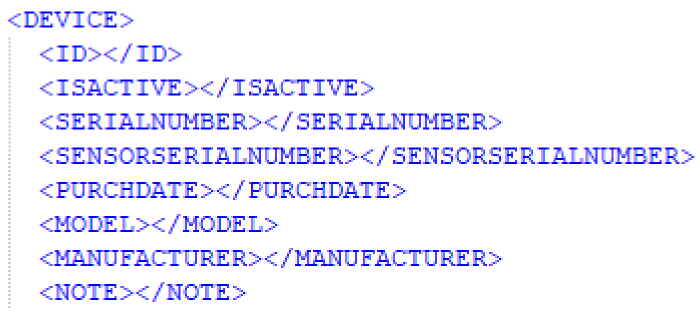
In this chapter most of the choices of technologies made when developing will be explained further.

3.5.1 XML

Choosing XML to transport data was easy. It was chosen since a schema already existed for validating data in XMLs. This schema could validate the entire instrument structure with hundreds of fields. Remaking this XML Schema using another technology would have required a lot of time. Since the schema is unable to validate anything but XMLs, the choice was clear. It was also chosen because of the amount of experience using XMLs.

JavaScript Object Notation (JSON) could have been an alternative to XML. Since the instrument structure often can have a lot of empty fields, it can lead to the size of the XML to be big even though not many instruments are added. (Code e.g. 13) For example when calibrating an electrical indicator, you are not likely going to add any data to the fields related to weighing instrument. This would be solved by using JSON which works with key value pairs and will only show the fields used natively.

Code e.g. 13



```
<DEVICE>
  <ID></ID>
  <ISACTIVE></ISACTIVE>
  <SERIALNUMBER></SERIALNUMBER>
  <SENSORSERIALNUMBER></SENSORSERIALNUMBER>
  <PURCHDATE></PURCHDATE>
  <MODEL></MODEL>
  <MANUFACTURER></MANUFACTURER>
  <NOTE></NOTE>
```

3.5.2 XML Schema

Data validation is important when handling large amounts of data, especially when inserting the data into a database. For data validation XML Schema was chosen, because like mentioned before, an applicable XML Schema already existed. The XML Schema was considered valid for validating data, before inserting data into the CMX database. However, some other technologies were considered.

DTD was considered because of its wide support in applications using XMLs, but dismissed because XML Schema was supported in every technology used in Data Loader anyway. DTD offered no other benefits over XML Schema. RELAX NG could also have been used, and would have been a more user-friendly option. It was however, dismissed because it lacked some validation features compared to XML Schema.

Validation could also have been done in the spreadsheet, but this would have meant rewriting all the validation rules to the spreadsheet, which would have been considerably slower than just converting the spreadsheet to an XML and then using an XML Schema to validate it.

3.5.3 .NET

The choice of using .NET was motivated by a couple of reasons. First, .NET is used in CMX which the tool will be a part of. Using the same technology would almost guarantee compatibility between the two software out of the box. It was also easy to find fitting components in .NET to satisfy the requirements for the solution. This meant that development of the application could be started quickly.

Considering the environment in R&D, investments have been made to support development using .NET by getting the newest IDE supporting it, Microsoft Visual Studio 2017. Of course, there are a lot of good free to use IDEs that support different technologies that could achieve the same result. But since a great IDE was available, working with it was chosen. Experience using these technologies and the environment also supported the choice. .NET is also used extensively throughout Beamex.

Visual studio supports a good mix of languages in the same solution and other technologies apart from .NET could have been used. JavaScript would require an adapter to be used with the CMX object model. One of the requirements was that the solution would not require any third-party software to work. This leaves out Java which would require Java Runtime Environment to run the application developed in Java.

Choosing the LINQ component in .NET was easy. The previous component used to convert the spreadsheet to XML could only convert it to a flat structure, and since this would be horrible for the user experience, it could not be used.

3.6 Bugs

3.6.1 No validation

The demo software used, came with a spreadsheet and a schema that were correctly formatted. But, when using the schema developed for Data Loader, no errors were found. Not even when deliberately adding invalid data to the spreadsheet. The only way to solve this was to add the namespace tag to every element in the code. First the tag was added only to the root element, but the element below that would automatically get an empty attribute. So, the tag was added to every single element in the code (Code e.g. 10). This may be a .NET specific requirement. The tag was easily added with find and replace. This was a particularly hard bug to find considering that the namespace attribute is not a requirement according to the XML specification. So, the validator used for most debugging never said that the schema was badly formed.

3.6.2 Adding multiple root elements

After putting the entire structure in a loop, the XML created would only use data from the last instrument in the spreadsheet. This is because the XML can only have one root element. So, every time it looped through the structure, every element was replaced. This was solved by first creating the root element and then adding instruments in the loop. (Code e.g. 12)

3.6.3 Validation error for empty elements

The first time Data Loader worked with the schema for Data Loader, a lot of validation errors appeared. Upon closer inspection the errors were usually related to the data being in the wrong format. But no data had been entered. So, the validation with the schema did not accept empty elements. This can be fixed in multiple ways. One solution is removing empty elements in the XML somehow. The solution was to specify a default value “” for elements. Back to find and replace.

4 Result

The result is an application that parses a spreadsheet of a certain structure and creates an XML from it, and then validates the data in the XML. This allows the customers to check their data for errors that may make their data incompatible with CMX. The validation is done using a schema that is supplied with the application. The schema represents the entire instrument structure and all required validation to ensure compatibility with CMX. With the related template, the customer can add data into the spreadsheet, import it to the tool and check if the data is valid. The template will help customers get their data in the correct format.

5 Discussion

Considering the result, development of Data Loader is nowhere near done. The future of Data Loader will be discussed in this chapter.

5.1 Future of Data Loader

In the future a user interface will have to be developed as specified by the requirement. It will have to look like a Beamex product, and research and development regarding user friendliness will have to be done.

The application is not integrated to CMX yet either which will have to be done soon. For this the team working on CMX will be available for developing and testing Data Loader. Some people at Beamex on summer job in the R&D department will also be directly involved with Data Loader.

Data Loader will be a part of CMX and will be released with the next major version of CMX.

5.1.1 Software coupling

Currently in the application, if the schema changes, changes will have to be made to the application as well. This is not optimal, for example if you forget to change one of them, they are going to be incompatible. Luckily version numbering will be implemented to mitigate this problem. However, there may be better ways to solve this, for example generating a schema from the code, or vice versa. Generally, when writing software, you want to avoid having to make changes in multiple places for the same change. This phenomenon is called loose coupling. Data Loader and CMX are loose coupled.

5.1.2 Static code

The elements in the code that create the XML from data in the spreadsheet are statically assigned to a column in a spreadsheet. This is not a good solution considering if the user accidentally adds a column first, the entire spreadsheet will be shifted one column to the right and will cause validation errors for most fields when the element is assigned to the wrong column.

This could be solved by linking the elements to the first cell in the column which contains a description of the field. So, for example the name element is assigned to a column where the first cell contains name. The descriptions are unique for every column used. So, every element would look for the column with a specific text in the first cell that corresponds to the element name and then use that column as a reference for every row under that column.

Making the bindings dynamic would be a huge improvement to the application as it would also allow customers to specify data in the order that they want. For example, if a customer is using mostly electrical equipment in calibration they could just specify the fields related to electrical equipment and Data Loader would find the appropriate column and link it to the appropriate element in the instrument structure.

This however requires additional parsing of text in cells and could affect performance of the application negatively.

5.1.3 Dynamic elements

Another criticism of the application is the fact that every element is added to the XML even though the element may be empty. Not adding empty elements could potentially give a huge performance improvement. It would also increase the readability of the XML.

5.1.4 Performance test

One other important thing that should be done before releasing this kind of software is to test its performance. The issue in the first place was that it was time consuming to add data manually to CMX, especially if the number of rows of data go up to the tens of thousands. Data Loader is obviously not going to perform the same using one row or tens of thousands, so a test spreadsheet with tens of thousands of rows will have to be made to test the performance of Data Loader.

6 References

Atwood, J., 2008. *XML: The Angle Bracket Tax.* [Online]
Available at: https://blog.codinghorror.com/xml-the-angle-bracket-tax/
[Accessed 9 May 2018].

Beamex Oy Ab, 2018. *Integrated Calibration Solution.* [Online]
Available at: https://www.beamex.com/solutions/integrated-calibration-solution/
[Accessed 8 May 2018].

Behera, C. K., 2017. *C# Corner.* [Online]
Available at: https://www.c-sharpcorner.com/article/validation-excel-data-in-c-sharp/
[Accessed 12 April 2018].

Bohring, H. & Auer, S., 2005. *UNIVERSITÄT LEIPZIG.* [Online]
Available at: http://www.informatik.uni-leipzig.de/~auer/publication/xml2owl.pdf
[Accessed 9 May 2018].

Box, D. & Hejlsberg, A., 2007. *LINQ: .NET Language-Integrated Query.* [Online]
Available at: https://msdn.microsoft.com/en-us/library/bb308959.aspx
[Accessed 13 May 2018].

Briganti, D., 2018. *W3C XML Schema (XSD) Validation online.* [Online]
Available at: http://www.utilities-online.info/xsdvalidation/
[Accessed 11 May 2018].

Champion, M., 2007. *.NET Language-Integrated Query for XML Data.* [Online]
Available at: https://msdn.microsoft.com/en-in/library/bb308960(en-us).aspx
[Accessed 17 May 2018].

Microsoft, 2018. *Get started with the .NET Framework.* [Online]
Available at: https://docs.microsoft.com/en-us/dotnet/framework/get-started/
[Accessed 8 May 2018].

Microsoft, 2018. *OLE DB Programmer's Guide.* [Online]
Available at: https://msdn.microsoft.com/en-us/library/ms713643(v=vs.85).aspx
[Accessed 9 May 2018].

Microsoft, 2018. *System.Xml.Linq Namespace.* [Online]
Available at: https://msdn.microsoft.com/en-us/library/system.xml.linq(v=vs.110).aspx
[Accessed 9 May 2018].

Mowry, A., 2015. *DigitalCommons@WayneState.* [Online]
Available at:
https://digitalcommons.wayne.edu/cgi/viewcontent.cgi?article=1103&context=libsp
[Accessed 9 May 2018].

Storset, L. A., 2007. *File:Common Language Runtime diagram.svg.* [Online]
Available at:
https://commons.wikimedia.org/wiki/File:Common_Language_Runtime_diagram.svg
[Accessed 17 May 2018].

W3C, 2008. *Extensible Markup Language (XML) 1.0 (Fifth Edition) W3C Recommendation.* [Online]
Available at: https://www.w3.org/TR/REC-xml/
[Accessed 9 May 2018].

Walsh, N., 1998. *A Technical Introduction to XML.* [Online]
Available at: https://www.xml.com/pub/a/98/10/guide0.html
[Accessed 9 May 2018].

van der Vlist, E., 2001. *Comparing XML Schema Languages.* [Online]
Available at: http://www.xml.com/pub/a/2001/12/12/schemacompare.html
[Accessed 10 May 2018].