NTNU
Norwegian University of Science and Technology

File Based Input and Results Database for «Focus Konstruksjon» Structural Analysis Software

Kristoffer Markus Kopperud

Master of Science in Engineering and ICT
Submission date: June 2017
Supervisor: Bjørn Haugen, MTP
Co-supervisor: Erik Aasmundrud, Focus Software

Norwegian University of Science and Technology
Department of Mechanical and Industrial Engineering

File based input and result database for Focus Konstruksjon structural analysis software

Kristoffer Markus Kopperud

June 2017

MASTER THESIS

Department of Mechanical and Industrial Engineering

Master's Degree Programme in Engineering and ICT

Norwegian University of Science and Technology

Supervisor 1: Professor Bjørn Haugen

Supervisor 2: Project owner Erik Aasmundrud

Preface

This masters thesis was carried out in the spring of 2017 at the Department of Mechanical and Industrial Engineering (MTP) through the master's degree programme in Engineering and ICT. This thesis has been written partly at Focus Software's office at Billingstad outside of Oslo, and at NTNU. Implementing a file based system for Focus Software's product, Focus Konstruksjon, is the aim of this thesis.

Before beginning this thesis, a pre-thesis project was written in the autumn of 2016. The deliverable of this project was a research of file formats that could suit Focus Konstruksjon. The results from this project is embedded into this thesis.

The file based system introduces a new way of transferring data through the software, by using files. This implementation is a backend solution, where the changes are done in the core of the software. This makes illustrating the implemented changes challenging. However, the thesis is written to give the reader the insight needed to understand the file based system.

Trondheim, 2017-06-11

Kristoffer M. Kopperud

Kristoffer Markus Kopperud

Acknowledgment

First and foremost I would like to thank my two supervisors, Bjørn Haugen and Erik Aasmundrud for their invaluable help and advice during the work on this thesis. I learned a lot through my work, which will be important for me in the future.

I would also like to thank Focus Software for letting me write and develop the thesis at their office at Billingstad. All employees have been friendly, helpfully and fun to be around. Thank you for giving me this opportunity.

Thanks to all inhabitants of my office, 230, and the atmosphere that has both been of great help and great distraction for my thesis. A special thanks to Are Fetveit for our motivational ice hockey games.

I'd also like to thank my family for their support this spring.

Last, but definitely not least, I would like to thank my beloved fiancée Camilla Lucia Ramse for all her invaluable help, encouragements and patience with me during the writing of my thesis. Thank you very much!

Kristoffer Markus Kopperud

Summary and Conclusions

To assure that Focus Software continues to be competitive in the construction market with their software Focus Konstruksjon, the software had to undergo changes related to data processing. The users are today modelling and simulating complex constructions without the ability to store their results. This thesis identifies the implementation of a file based system as the solution to this challenge.

The master thesis began with researching the field for both pre-processing and post-processing file formats, respectively the input and the output format for the file based system. The evaluation of these findings resulted in the development of a system which utilizes the Usfos and the VTK format. The system development process included code base familiarizing, prototyping, a test driven development approach, testing of the system, and an implementation of the file based system into Focus Konstruksjon.

The file based system now allows users to store their results, enabling visualization of their result databases without having to re-run the structural analysis. This generalization of data processing is beneficial for Focus Konstruksjon in future development, testing and maintenance, because of looser couplings and a more readable data structure. The system is implemented for linear structural analysis, and will in the future be integrated into other structural analyses provided by Focus Konstruksjon.

The deliverables of this thesis is the written thesis and its research, the code of the file based system, tests, documentation and recommendations for future work of Focus Konstruksjon.

Sammendrag og Konklusjon

For å sikre at Focus Software forblir konkurransedyktig i byggbransjen, må deres programvare Focus Konstruksjon gjennomgå endringer relatert til databehandling. Brukerne modulerer og simulerer komplekse konstruksjoner uten å kunne lagre simuleringsresultatene i dagens programvare. Denne oppgaven identifiserer implementering av et filbasert system som løsning på denne utfordringen.

Masteroppgaven begynte med en studie av feltet for pre-prosesserings- og post-prosesseringsfilformater, henholdsvis inputfilen og outputfilen for det filbaserte systemet. Evalueringen av filformatene som ble funnet i den prosessen, resulterte i en beslutning om å benytte Usfos-formatet og VTK-formatet i det nye systemet. Utviklingsprosessen omhandlet det å gjøre seg kjent med kodebasen, utvikle prototyper, benytte testdreven utvikling, testing av systemet og implementering av det filbaserte systemet i Focus Konstruksjon.

Det filbaserte systemet muliggjør nå lagring av brukernes resultater, som gir til muligheten til å visualisere deres resultat database uten å måtte gjennomføre analysene på nytt. Denne generaliseringen av programvaren er en fordel for Focus Konstruksjon med tanke på videre utvikling, testing og vedlikehold. Systemet er implementert for lineære analyser og vil i videre arbeid bli implementert for de resterende analysene i Focus Konstruksjon.

Denne masteroppgavens leveranser er studiet av filformater og funn som er skrevet i oppgaven, koden til det filbaserte systemet, tester, dokumentasjonen og anbefalinger til videre arbeid av Focus Konstruksjon.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Focus Konstruksjon is a software program for structural analysis, made by Focus Software. The software is built for creating all from small to big complex constructions. Since Focus Konstruksjon primarily are targeting customers in Norway, they can provide a good customer support system and specialize the standards and requirements needed for a Norwegian environment. As a software developing company, it is important for Focus Software to follow the market trends, and they are always looking to be competitive. This thesis is focusing on improving Focus Konstruksjon, which is one of their products.

Problem Formulation

Focus Software wants to be competitive. Therefore, they are looking for a solution to make their product easier to work and collaborate with. One of the main challenges is that the product does not support storing the result database of a FEM-structural analysis. This implies that when a user has viewed the results, and closed or moved on to other parts of the program, it is not possible to review the results since the result database is erased from the computer memory. The constrains, conditions, loads and materials on a construction are still available. However, the analysis must be run again to view the results of the simulation. Solving this problem requires the result database being stored to a file, instead of the data being temporarily stored in the memory of the computer.

Focus Konstruksjon uses a simulation software called OOCfem. This simulation software is an integrated part of Focus Konstruksjon, and during the transfer of data between the two software programs no data is stored. This means that the data sent from Focus Konstruksjon software, which implies 3D models with geometric data, constrains, conditions, materials and loads, are temporarily stored in the memory. OOCfem, the simulation software, gathers the necessary data from the memory as an input to the simulation program, by using an OOCfem Application Programming Interface (API). In simpler term,s the Focus Konstruksjon's internal data structure of a model is converted to OOCfem's internal data structure, without any data stored to a disk or a file. Similarly, after the simulation is done, OOCfem stores the output database temporarily in the memory. Focus Konstruksjon gathers the data from the memory of the computer, then the result database is shown in the visualization part of the Focus Konstruksjon. See Figure 1.1 for an illustration. This dataflow makes the process very quick, but at the same time there is no data stored outside of the process.

As a competitive software on the construction market, the challenge above might at

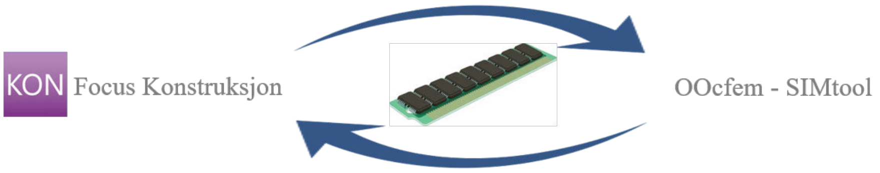


Figure 1.1: Illustrating today's data transfer system

first seem as a problem that should have been solved at an earlier stage. The reason the challenge is still present today is because of Focus Konstruksjons history. From the beginning, the software solved small simulation problems like singe beam constructions. This required little resources and time. When the software developed to handle buildings and other constructions, users were more interested in new features than having the software run faster. However, when the user now spends over twelve hours to simulate a building and every time the user closes Focus Konstruksjon, a simulation has to be re-run to visualize the results again. It has now become a problem for the users, and Focus Software wants to handle this challenge and solve the problem.

To solve the problem, a file based system must be introduced. When model and simulation data is transferred from Focus Konstruksjon, the data should be stored in a data file, which is seen as external storing. Similarly, the result database should be stored in a data file. To be able to store data in a file, it is important that the file has a structure which organizes the data. The structure lays the foundation for how complex the new file based system will be, making the format an essential part for the new system. The research done in this thesis shows that there are many ways to store data to a file and many different standards are used to store 3D data today.

This thesis will first look at what file formats that might be candidates for Focus Konstruksjon. It will look at file formats for the pre-processing phase as the input file and the post-processing phase as the output file. Further, the thesis will look at different aspects of the structures in each file format. Why are they suitable for a Finite Element Model (FEM-model) and why are they suitable for Focus Konstruksjon? The input data and the output database will hold different information and be structured differently. That is the reason that this thesis evaluates two separate files for the two phases of transferring data in the new file based system, from Focus Konstruksjon to OOCfem and back again. See the process illustrated in figure 1.2.

To successfully integrate a file based system for both Focus Konstruksjon and OOCfem, developing requirements to separate formats in the pre-processing phase and the post-processing phase is important. These requirements should focus on file formats qualities to be able to distinguish formats for complete a proper evaluation. An evaluation of the formats found during research will be presented, and a new self-defined file formats will be introduced. Further, the prioritized file formats will be implemented on top-level (prototyping), to identify the strengths and the weaknesses, before choosing

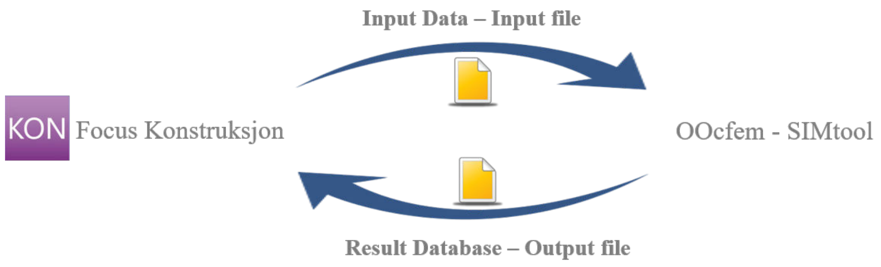


Figure 1.2: Illustrating the preferred future data transfer system

the preferred format. Examples of strengths and weaknesses of the formats, can be challenges met during implementation. Such as, how costly the approach of implementation seems or what assisting tools are available and what work have been done with these formats earlier.

This thesis project only allowed a short term engagement in Focus Konstruksjon, which required a plan on how the acquisition should be handled. The typical documentation from a software project was planned, such as code quality, documentation and tight communication thoughtout the project, making the acquisition even better. Seamlessly tests for the code was decided be developed to help understand and control the functionality of the file based system. Especially when the Focus Konstruksjon team are to further develop and maintain the system in the future, the tests will be helpful. Therefore, this thesis will focus on the benefits of tests and how tests are written, together with a chapter in this thesis describing test driven development.

The development of the file based system began after choosing the file format for both the pre- and post-processing phase. This thesis is thus divided into the code development part and the part describing the implementation process. The deliverables of this thesis is therefore the research leading up to implementing two file formats and then briefly describing the key elements in the implementation process. Other deliverables that are not presented in this thesis, but is delivered to the developer teams of both Focus Konstruksjon and OOCfem, are the code, tests, file format documentation and knowledge shared during the process. A discussion of the findings and future work finalizes this thesis.

In summary, the objectives for this thesis are:

1. Developing requirements for the file-structures and for parsers and writers.
2. Researching what types of file-structures Focus Konstruksjon can utilize.
3. A comparison of existing and self-defined formats.
4. Prototyping prioritized formats to identify strengths and weaknesses.
5. Choosing file formats after familiarizing with Focus Konstruksjon and OOCfem code bases.
6. Present test driven development and how it have been used during the thesis.
7. Key elements in the implementation process.

1.2 Approach

This thesis differs from other science theses, because it is not done to support hypothesis with theories in a university research standard approach. This thesis seeks to get an overview of the existing technologies, file-structures and the current approach of storing pre- and post-processing data. The next goal is to further develop the new chosen file based system as a software development project. Followed by an implementation of the system in Focus Konstruksjon, to help improving a competitive solution of Focus Konstruksjon, for Focus Software.

1.3 Structure of the thesis

The structure of this thesis is based on the natural development of a software project. Where the actual code development is documented in the code, which leads to that not all deliverables are presented in this thesis text. However, deliverables could be found in the code it self. Especially since this is a backend solution, there are not many sufficient ways to illustrate the work. Therefore, the reader is presented key elements of the development and the implementation process. The reader of this thesis is expected to have basic knowledge of software development, FEM and CEA modeling.

Chapter 2

General Review and Overview

To gain an overview of the existing research, technologies, and development of relevant file formats, a study of the existing literature and documentation was conducted. The purpose of this chapter is to frame the extent of the current research and documentation on different file formats for the pre-processing and post-processing phase. In general, the pre-processing phase is the process where a CAD model is converted to a FEM model, and all the required data to run an analysis are added to the model. The pre-process data is the described FEM model data, which is referred to as the input file in this thesis. The post-processing phase is the phase where result data from a simulation is sent to be visualized as results of a simulation, and is referred to as the output file.

A satisfactory file based system needs to reflect the requirements from all stakeholders, Focus Konstruksjon and OOCfem. Since OOCfem is developed by another actor, the requirements need to be considered on the same level as for the developers of Focus Konstruksjon. From the information found in the requirements, a general overview of file formats, for the pre-processing and post-processing file formats, and the structures will be created. This chapter will also give an insight into what technologies are relevant for this thesis and what type of tools are used.

2.1 Requirements

After a meeting on the 19th of September at Focus Software's office at Billingstad in Oslo, the basis of this thesis was decided. The main focus of the update of Focus Konstruksjon was to make the software able to store the results after running an analysis of a model. However, there were other requirements discussed in this meeting and in later meetings. The requirements are covered in the functional and non-functional requirement sub-chapters below:

2.1.1 Functional Requirements

- First priority functional requirements:
  - FR1: Focus Konstruksjon should write and store data from the pre-processing phase to an input file.
  - FR2: OOCfem should read input data from the pre-processing phase from an input file.
  - FR3: OOCfem should write and store data from the post-processing phase to an output file.
  - FR4: Focus Konstruksjon should read output data from the post-processing phase from an output file.
  - FR5: Both the input file and output file should be stored in such a way that it can easily be used by Focus Konstruksjon and OOCfem.
  - FR6: The input file implementation should have correspondent tests, to verify that correct data is written and read.
  - FR7: The output file implementation should have correspondent tests, to verify that correct data is written and read.
- Second priority functional requirements:
  - FR8: An input file format should have a file structure which can be read by a developer or user.

  - FR9: Input file data should have the ability to be changed directly in the file by a text editor.
  - FR10: An input file format should be an ASCII file.
  - FR11: Output file should be possible to read, not necessarily directly.

- Third priority functional requirements:
  - FR12: A post-processing output file format should have a file structure which can be read by a third-party 3D-visualization program, specialized on displaying results from a finite element analysis.

2.1.2 Non-Functional Requirements

- First priority non-functional requirements:
  - Performance
    - NFR1: The file format system of Focus Konstruksjon should have as short response time as possible.
    - NFR2: The file format system of Focus Konstruksjon should use as little memory as possible.
  - Documentation
    - NFR3: The file format system of Focus Konstruksjon should be well-documented.
    - NFR4: Both file formats should be well-documented.
  - Open Sourced
    - NFR5: The file format system of Focus Konstruksjon should be open sourced. This implies that Focus Software do not want expenses on licenses and software.
- Sound priority non-functional requirements:
  - Maintainability

* NFR6: The file format system use of external interfaces must be without any restrictions and risk for interfaces no longer being provided or developed.

2.2 File Format Findings

The findings are divided into three categories; File Formats in General, Pre-processing and Post-processing.

2.2.1 File Formats in General

Today, there are some software for CAD- and CAE-modeling holding a large part of the market share [1]. These software programs' developed systems to store pre-processing and post-processing data files, and some of these programs have file formats that again can be read by several other software programs. Open sourced formats however, which is a requirement for this thesis, are often more difficult to research. Researching different formats showed that they often do not meet the complexity required for a pre-processing format from Focus Konstruksjon, and documentation is often difficult to find. The maintainability, continuous development and usage, are not that common for open sourced formats.

Well-defined and well-documented are two definitions that will be used during the thesis. First, a file format could be well-defined and well-documented in terms of how the data structure is presented and how simple it is to get an understanding of the file structure with the available documentation. Second, parsers and writers as software interfaces must be well-defined and well-documented in order to be integrated into a software efficiently. The definitions is used for both the formats and for the software handling the formats. A parser is an element within programming which reads input data and creates a data structure [2]. Therefore, the thesis will use about both parsers and writers as what is required of code development when analyzing file formats for reading and writing data.

The research of this thesis discovered that many pre-processing file formats are developed to fit the simulation software used in the pre-process phase. Implementing a file format specialized for another simulation program would be difficult to use directly. Since Focus Konstruksjon uses OOCfem, implementing a pre-processing file format will be difficult without making a specified parser and writer, and without doing changes to the format.

Requirements say that the pre-processing file should be ASCII. ASCII, also known as American Standard Code for Information Interchange, represents text in computers. Today most text represented on devices is based on ASCII and not binary files [3]. The reason for using ASCII files is to give the opportunity to examine and change details in the file. This is an advantage for testing, controlling and future development of both Focus Konstruksjon and OOCfem. The potential positive side of choosing a binary format is that it is more likely to meet the performance requirement better than an ASCII file [4]. The research done revealed that the pre-processing files are often ASCII text files, and binary solutions are more often used in post-processing file formats.

Open sourced is a non-functional requirement from Focus Software. Research found that there are different ways to classify open source software GPL, LGPL and BSD [5] are the ones found relevant when evaluating file formats for this thesis. These abbreviations stands for, GNU General Public License, GNU Lesser General Public License and Berkeley Software Distribution.

Any LGPL licensed open source software that is used by another software or application will allow the program or application to be sold without having to release the source code of the program or application.
On the other hand, if a GPL licensed open source software is used, it requires the program or application to be licensed under the GLP. This requires all software using the GPL licensed software to be open sourced as well. Focus Konstruksjon could then not be an actor on the commercial market, since Focus Software must have had released some of the software as open sourced. These are both GNU licenses which have li-

censed that the software is open sourced, and are developed in a community. The users of the software are required (GPL) to contribute with their own software, or it is recommended (LGPL) that they share their own software.
The BSD license has a license which is based on a user receiving the library once, and if there are any problems there is no developer community offering help, it is now the problem of the developer.

2.2.2 Pre-Processing

The pre-processing phase is described by modelled Computer Aided Design(CAD)-data. CAD files are a representation of a 3D model design, which further is used by other software. A mesh generator then creates mesh data, and together with boundary conditions, constrains and loads. After meshing, the model (normally called the FEM-model) represents the finite element data which is sent to a simulation tool for analysis. In this thesis the FEM-model will be generated by Focus Konstruksjon and then sent to OOCfem for analysis.

Through research of the pre-processing input files, it became apparent that CAD-files could often be read by a number of CAD software programs. In addition, there are also converters available, which enables several possibilities for reading CAD-files [6]. However, there were many different solutions to the structure of the data, e.g. meshes, boundary conditions and load data. This makes the FEM-file different and not that uniform. The consequence of this is that the CAD software easily could exchange models. On the other hand, there are few standard ways to build up a FEM-file while following software specific steps of the pre-processing phase. Even if there are some certain rules when it comes to writing FEM elements to a file. Some FEM files intended on structural analysis have focus on structuring nodes and elements, and loads connected to those. Where on the other hand, FEM files intended on fluids or micro mechanism, focuses on FEM-mesh representation such as triad mesh elements.

What kind of operations conducted during the pre-processing phase and the simulation process was found to be individual for different programs. Some programs stored the loads for each element and node, and some stored the loads for the models and the loads were later computed in the simulation part. Different simulation programs have different requirements for a file or the data structure. The modelling software and the simulation software is often tightly developed together over the years with the intention to make a seamless product. This results in a data structure highly qualified for exactly one pre-process and tightly coupled software, which makes it less qualified for other FEM programs.

Some converters were discovered through researching ways to store FEM files. Jotne IT have converters from two formats; NASTRAN and Abaqus [7]. This is a complete solution where the users are required to pay for the product. There are also some open sourced efforts. The general findings are: “Although it took us a lot of efforts to roll this out, many features are still missing” [8]. There are some conversion programs developed by professors for special cases, and sometimes narrow-focused software. These are not general enough to deliver a trustworthy and accurate result when converting FEM-data. This supports the findings of pre-processing data often being specific, and that a standard format to use directly will be difficult to find.

If Focus Konstruksjon has its own structure of data, how could it then be possible to use an existing file format? The answer is found in the documentation and developing of parsers and writers. Focus Konstruksjon follows the data structure of a 3D model build up, and after creating a FEM-model the structure contains nodes, elements, load groups, constrains etc. Parsers and writers must therefore be developed to utilize the structure of a FEM format. However, since the model software and the simulation software are seamlessly developed together, special model cases are already integrated. As mentioned, research found that an input file cannot be implemented directly. Still, the structure of nodes, elements and load groups are already in place. A file format that is close to how Focus Konstruksjon and OOCfem communicates, is a good opportunity to cut development costs in terms of less development of parsers and writers. Further, a specialized parser and writer must be developed to match the already seamless integration between Focus Konstruksjon and OOCfem. This implies that it is important to find a file format which suites Focus Konstruksjon well, which also have low development costs.

The challenge is to find a file format that is well documented, both as a format and for parsers and writers, creating a balance between using the existing format and the developing cost. The research will demonstrate if it is possible to find that balance. A self-defined file will also be evaluated as an input file, and is especially important to research if the balance mentioned is difficult to find. Developing a self-defined format could lead to high development costs in the beginning, although this will lead to high quality seamless integration.

2.2.3 Post-Processing

The post-processing phase is described by the result database from the simulation being imported by a software that does visualizations of result data. Usually, the visualization software is the same software that exported the FEM-model in the pre-processing phase. The purpose of the process is to display the results of the analysis. This implies that together with the usual 3D data, the post-processing result database contains displacement, stresses, momentums, and other result data for structural analysis.

Research showed that there are different file formats that Focus Konstruksjon could use. Compared to the pre-processing phase, the results from the simulation analysis are in general not as specialized for one single software as for the pre-processing phase. The open source license challenge faced in the pre-processing phase is also present in post-processing. Findings also showed that compared to pre-processing formats, there are not as many file formats needing to be evaluated. The file formats found relevant for Focus Konstruksjon, serve the same purpose and the formats are similar, more than what is seen in the pre-processing formats.

A requirement influencing file formats for the post-processing phase, is that the file format should be compatible with a third-party visualization software. Even if it is a third priority requirement, the benefit of visualizing the result database in other software is that a developer has more than one way to look at the results from simulations. Results visualized in other engineering software might support different views and features that could have favourable influence on how Focus Konstruksjon could be developed in the future. Research of different visualization programs showed that VisIt, Paraview or Techplot are tools that could meet the requirement of third-part 3D-visualization program. See chapter 3 for research where Paraview was found as the recommended program.

2.3 Technical Overview

2.3.1 Programming Languages

From the introduction in chapter 1 it is known that to improve the pre-and post-processing phase for Focus Konstruksjon, both software programs, Focus Konstruksjon and OOCfem, need to be analysed and developed. Since the challenge was to improve the software and not build it from the beginning, the two software already have considerable code bases. Focus Konstruksjon is written in c#. OOCfem is written in C++, with managed C++ code. These languages are well used and well-documented, and are one of the top five used programming languages in the world [9].

C++ is developed as an object orientated language, based on a program language called C. The C language is seen as a fast running language. This is because it takes a short time compiling the code, which is close to machine code. C++ is broadly used today [10], especially in system development, as a result of height performance and flexibility. Since OOCfem is a software developed for structural analysis, C++ was a natural choice because of height performance and good documentation.

Microsoft developed a .NET framework at the end of 1990, which offers a big class library; Framework Class Library (FCL)[11]. This framework was developed for users to combine their source code with the framework library, for increasing productivity. By including the class library, the .NET framework, it becomes possible to combine different high-level programming languages such as C++, C, Python and c#. The .NET framework runs on a virtual machine where it is easy to combine the different languages, it is called the Common Language Interface Runtime. When code is depending on the Common Language Interface Runtime machine to run, it is referred to as managed code. During the process of developing the FCL library a language, later called c#, was created. This means that c# is the language that is closest to the virtual machine of the .NET framework library and is the main managed language in the .NET framework Runtime machine.

Unmanaged code is code which can be compiled directly to machine code, and managed code is code that needs the Common Language Interface Runtime machine as an intermediary part to compile and run code. Usually C++ code is compiled directly to machine code. However, managed C++ code is code used to connect the c# language and C++. In Focus Konstruksjon's case, the managed C++ code is the code that binds the software and OOCfem together. The purpose of using managed code in Focus Konstruksjon is also to prevent an unstable application [12]. Be aware that Microsoft launched their definition on managed C++ code, and it is slightly different from managed and unmanaged code generally used in software development.

Focus Software wanted to see how F# programming language could be used in developing file formats in Focus Konstruksjon. F# is a programming language which encircle programming techniques such as imperative, object-oriented and functional programming, with a focus on the functional part. A functional language presents an opportunity to read and write files easily, compared to fully object-oriented languages [13]. During research it will be evaluated if F# should be used or not.

2.3.2 Development Environment

Microsoft Visual Studio is an integrated development environment (IDE) developed for developers on the .NET platform. In Visual Studio, the user has the opportunity to produce, run and debug both managed and native code. There are many versions of Visual Studio, all have support for debugging, syntax help, code auto complete functions and GUI-applications. Visual Studio also supports third-party plugins, such as plugins for

GitHub, and installation of different packages (NuGet). Developing a file based system in Focus Konstruksjon and OOCfem which uses respectively c#, managed C++ and C++, using Visual Studio as the preferred IDE is an evident conclusion.

2.4 Summary of the review

From the general review conducted in this chapter it is believed that an introductory overview has been achieved. The requirements listed in the chapter give a satisfactory overview of what should be the main focus area in developing the file based system. By summarizing the two processing phases it is clearer what factors should be evaluated when selecting file formats. The chapter have revealed that there might not be as many file format options relevant for this thesis as first expected, especially regarding the output format. Therefore, the formats chosen must be properly evaluated. When evaluating it is important to be certain of what requirements could eventually be a “deal breaker” for the suggested file format.

Chapter 3

Evaluating File Formats

This chapter will evaluate different file formats found during research. Some file formats will not meet the requirements presented in chapter 2. They are however still evaluated. In that way, the reader is given an insight on what file formats that could be relevant for Focus Konstruksjon in the future, or not worth considering when further developing the software.
Focus Software should also know what file formats are well-supported by other software programs, especially third-party programs. Focus Konstruksjon's users could have a seamless experience if other software supported their Focus Konstruksjon generated formats. The chapter also provides a summary of the results in the form of tables, and a final evaluation of the formats.

3.1 Pre-processing – Input file

As mentioned in chapter 2.2.2 the pre-processing phase has a variety of formats that are well-defined and well-documented and formats that are developed for special cases. For a well-defined and well-documented file format, changing and develop Focus Konstruksjon and OOCfem is necessary to adapt to this sort of file formats. For a file format with low complexity and little documentation, development of the format to today's version of Focus Konstruksjon and OOCfem is necessary. This illustrates two different consequences when choosing a format.

To get an understanding of what data structures are preferred in the new file based system, an investigation of the current data structures where preformed. The findings where that Focus Konstruksjon and OOCfem operated with FEM structural elements, such as beams. This differs from other specific FEM data structures who are focusing on the geometrical representations of mesh, such as triads.

The technical structural requirements of the file format, such as FEM elements, are researched to be similar for most of the available pre-process phase formats. All formats evaluated in this chapter have a structure which meets the data structure requirements of OOCfem. At least to the extent of possibly being developed as an input file for OOCfem. After conferring with the supervisors, it was decided that the formats should be evaluated after requirements closer to a business case approach. Where open source, documentation and performance are important issues. The input files will not contain as much data as an output file, therefore the performance requirement is somewhat reduced in the input file evaluations.

The formats evaluated are chosen because of their representation of the different pre-process phase file formats categories found during research. As well as they are researched to fit into a file based system in their category. Some formats are built by companies, some are built by research groups and some formats are chosen to present what formats that might be interesting for Focus Software in the future.

Some of the file formats researched and evaluated will be presented in the appendix chapter D. Although, these formats are still presented in the table seen in chapter 3.3.

3.1.1 List of File Formats Found

NASTRAN – Bulk files

The file format can be used as both a pre-and post-processing file. It meets the requirements set in chapter 2, with being well-documented and an ASCII file. Even if the format is readable as a text file, some will argue that the format is not easy to read and change. Where in other file formats white spaces are used to divide data, the bulk file does not have white spaces. It separates the data with counting digits [14]. This makes it difficult to read and change the format manually, and at the same time increases the risk for errors while users are changing data. The format has numerous FEM format file elements, called NASTRAN cards, which provides a complex format that can handle many different FEM elements. The Nastran bulk data files are more detailed evaluated in the post-processing chapter 3.2.

Compared to other file formats NASTRAN bulk data file is well-defined and well-documented. The difficulty with the format is that there are few open sourced parsers or writers. However, the theoretical documentation is available and open. Therefore, it is a combined open and not open sourced situation that has to be considered. Where the data structure of the format is documented, and associated software is not documented.

USFOS – UFO

Usfos is a structural analysis program for structures originally in the offshore sector. It is a simulation tool specialized on accidental load analysis. It has the opportunity to represent both pre- and post-processing. Since Focus Konstruksjon is based on structures, the two software's have similarities. Usfos are using a file format called UFO Structural File Format. It stands for a User-friendly structural file Format, which they claim they have special designed the file format for [15].The format is further developed and inspired from the Norwegian Veritas format; Sesam System Interface file [16]. This is a file format that is used for structural analysis in the Northern sea.

One of the reasons the UFO format is especially relevant for this thesis is that OOCfem already have used this file format in test phases earlier. With modification to the format to specify the structure to the data structure of OOCfem. The software has its own implemented classes to parse and write data in its owen format. This again lead to a specified file format for OOCfem called AME-fem input file format. It is created by

OOfem's developer and is well-documented [17]. A OOCfem specific implementation of the UFO format gives it an advantage compared to other formats, and should be considered as an important factor when choosing the preferred file format.

The developer of OOCfem says that the Usfos format was chosen was because together with co-developer, Tore Holmås. Since they have a well working relationship, further development of the format is in both interests, which is an advantage for OOCfem. Usfos also provides a lot of code and model examples for reading and writing the format. This was an essential part when choosing the format, OOCfem's developer mentioned. However, there are no official interfaces.

There are no complications with meeting the first priority requirements with this file format. The UFO format is an ASCII file format. This makes the file readable in text editors, which makes it possible for developers and users to read and do changes to the data directly in the file.

Figure 3.1: USFOS

The UFO format is well documented and developed over many years together with the USFOS software [18]. Together with the AME-file supplementation for OOCfem, it is well documented for Focus Konstruksjon's needs. The format data structure is open sourced, with no official open sourced software interfaces for parsing and writing the format. The format is a representation for FEM data, and does not have any representations for analysis types. If the format should be used as a pre-processing phase format, that has to be implemented. The thesis will from now mention the format as the Usfos format.

Gmsh – Mesh generator

During research, different mesh generators were researched. For a pre-processing point of view, a data file from a mesh generator would give the required geometric data of a file format for a FEM file. This format would have to be developed further with additional conditions, so that Focus Konstruksjon could utilize such format. With a good data structure documentation and well-defined parsers and writers, the disadvantage of the format only representing geometric data could be decreased.

The reason this mesh generator was relevant was the connections to other known file formats, such as VTK and NASTRAN [19]. A format that is compatible with a post-processing format, could be an advantage and reduce development and maintenance cost.

Gmsh have open sourced programs available. However, the open source restrictions are GPL and not LGPL. Focus Software Konstruksjon is a commercial product that will gain revenue on this library. Using the library is not possible without Focus Konstruksjon being open sourced GPL as well.

The research found this pre-processing mesh generator to be best suited to meet the requirements of a file based system, even with the open sourced restrictions on the associated software. Other mesh generators with near to equal solutions to Gmsh are not evaluated in the thesis. However, it is important for the reader to know that Gmsh and other mesh generators are suitable input formats, with developing the format to adapt to today's data structures. The license must nevertheless meet the open source requirements.

Figure 3.2: gmesh

STEP – AP209

The user's models and analysis data are desired to be exchanged between different software and platforms. Together with a tendency, which this thesis has mentioned earlier, of pre-processing data being customized to the isolated process of one simulation software. Some calls for a halt in this tendency. A new ISO standard extension of the STEP ISO standard where proposed, AP209, are tapping into this marked need [20].

Research found that there have been efforts to solve the same business need described above, by for example the format evaluated in appendix chapter D; femML. The problem with these earlier solutions is that there have not been supported from the large commercial companies and stakeholders. These companies want to keep their customers exclusively connected to their own software. Research showed that STEP is a broadly supported file format in CAD modeling. With a new standard FEM data representation, which originate from STEP, companies might be excepted to integrate this in the future. The AP209 also have the ability to not only handle FEM data, but also have the opportunity to handle post-processing data [21]. For Focus Software this could be an interesting file format for the future. It will be rapidly developed. Other users that might want to use Focus Konstruksjon's simulation software could do so through a AP209, if implemented.

The AP209 format meets the requirements of being able to change and read in a text editor. It is an ACSII file that for a user will be more difficult to read than other formats such as Usfos format, because of its structure. The format is well-defined and well-documented, with an open sourced documentation. Research showed that parsers and writers are to not too easy found nor accessed. A PHD-student at NTNU are writing his PHD on developing AP209. This could be a contact for later to learn more about the development of the format and marked influence for Focus Software.

VTK - Visualization ToolKit

A more thorough examination of the Visualization Toolkit (VTK) will be done in the post-processing output file chapter 3.2. The reason for this is that VTK is in general a software that can handle 3D computer graphics, image processing and visualizations. In 3D computer graphics, it is primarily focused on visualization of result databases for simulations. However, comments from VTK users shows that it is possible to use the unstructured grids in the VTK legacy format for a pre-processing phase data file.

There are no complications with meeting the first priority requirements with using the suggested technique. The VTK format could be written as a ASCII file, and is LGPL open sourced. This means it is a possible solution, however on the other hand feedback on this

Figure 3.3: VTK - Visualization ToolKit

VTK pre-processing technique is not unilaterally positive. “Rather than trying to shoehorn FEM data structures into VTK (which is not pretty), you can build adaptors from VTK to native FEM codes and data structures” [22]. The benefit of using VTK as the preferred output file format, is that an implementation in both the pre-processing and post-processing phase, would lead to reduced development and maintenance cost.

MED file format

The MED file format is a format specialized from a HDF5 standard [23]. It is used by a FEA solver called Salome. HDF5 is a relevant file format for the file based system, since the same format could be used for a post-processing file format. There are no complications with meeting the first priority requirements with this file format. However, the MED format is not an ASCII file format. But there are HDF5 frameworks that allows transformation of ASCII files to HDF5 files, and the other way around. Salome has a GNU LGPL licensed software, and HDF has no licenses attached to it. The format is well-documented.

Figure 3.4: Salome

Exodus II

This is a FEM data file format developed to represent mesh generated data, analysis data and visualization data, originally built on the NETCDF format. This is a set of format libraries that supports array-oriented arrays [24]. The Exodus II format has APIs for Fortran, C and C++. It also have available converters to other common FEM formats such as Abaqus .osb, NASTRAN and Universal File format (UF). It has a BSD open sourced license, which suites Focus Konstruksjon’s requirements well. Compared to other FEM formats it is more similar to the Gmsh file format than the Usfos format. Because the data structure are geometric representations, rather than structural FEM elements.

Exodus II meets the requirements set in the functional requirements. It has the possibility to convert from binary to ASCII format, which makes the format more flexible.

The format is, with the basis of the NETCDF format, linked to the post-processing format HDF5. That could possible lead to the same benefits as described for VTK.

3.1.2 Self-defined Format

After evaluating the existing file formats that could be relevant for Focus Konstruksjon, a weighting of benefits must be conducted. Either implementing an existing file format or a self-defined format that can use newer and faster developed parser technology found for formats such as json, yaml and XML. In this sub-chapter the thesis will look at the advantages and disadvantages with implementing a self-defined format.

First it is necessary to have an understanding of the different abstraction levels within Focus Konstruksjon. In focus Konstruksjon there is a wrapper that wraps the OOCfem API which is written in managed C++ code. This API is an interface on an API-class in OOCfem which is written as native C++ code. This API-class wraps the OOCfem operational class called cfem.cpp. The existing formats have a data structure where parsing the data and convert it to the internal data structure in OOCfem can be done on the lowest abstraction level. This means, implementing a format specific parser and implement it in the core of OOCfem, the operational class cfem.cpp. To benefit from implementing a self-defined format, the advantage must be found in different areas than the data structure in the format. Because if not, it would have been better to use already existing and documented formats, which also might have developed their own parser and writers. Therefore, the self-defined format should use areas other than the data structure of the formats to be beneficial. The abstraction level review shows that OOCfem already provides an API solution for Focus Konstruksjon. That could be where a self-defined format get an advantage. Since an existing format would be integrated and developed together with the operational cfem.cpp class. A seamless integration must be conducted, which again increase development costs, instead of using already existing code.

Therefore, the concept of a self-defined format is to use the already existing code and interfaces between Focus Konstruksjon and OOCfem. Since there are already an existing API which has methods calls containing the FEM model data. The methods call data (known as parameters) could instead of being written to the memory being stored as it is in a file. Without being stored in an external data structure, which the existing file formats provides. This implies that a data file will contain the same data as methods does in Focus Konstruksjon and are needed in OOCfem. The existing code in OOCfem that expects data from methods, which now is stored in the self-defined format, will not have to be developed to integrate the new format, by simply utilize the API. This will minimize the work on implementing a file format, since it is possible to reuse most of the seamless code that is already developed.

The disadvantage of a self-defined format is that it will contain more data than an existing file. Considering for example index lists, which is data used of the methods in the API for several cases must be written several times. This will not be necessary using existing formats. One node in existing FEM formats, that is represented as one line in the file format, holds information of coordinates, the index, translations etc. In the self-defined format nodes, it will represent one line per element in the lists of coordinates, indexes, translation etc. Potentially it could be expected $\Omega(xn)$, where n is the number of lines in an existing file format structure, and x is the number of arguments in the API method.

Although these challanges can decrease the chances of meeting the performance requirements. An effective data structure can solve it. If the data is structured as a dictionary where the method name is read first, the expected performance could then be $\Omega(m)$, where m is the number of methods called in one simulation from Focus Konstruksjon. Since the worst case is to read through the whole file, and each line represents a method call. That is an acceptable performance comparing with the already existing formats and the differences in development cost. In chapter 4, under prototype implementation, a more thorough description will be presented.

To maintain the potential benefits of a self-defined format, the format must have parsers and writers that are well-defined, broadly used, easy to implement and well-documented. In the always developing technical world, the research found an increasingly use of file formats for web applications. Research also found that the most common formats in that category are XML, Json and Yaml.

The XML format has been introduced in the femML evaluation earlier in appendix chapter D. The format is well-documented, although some will say that it is not an easy read format. Json is a JavaScript Object Notation, which is broadly used with well-defined interfaces, first developed as a replacement for XML in Ajax web applications [25]. Research showed that there are examples of how Json is used for storing data outside a web context, for example saving objects in C# to a Json file [26]. Json is a subset of Ymal. Yaml normally does not use brackets and indents like Json, but are built up by line and white spaces. For a FEM data structure, it might be more unclear what the different elements in the file are referring to in a method data representation. Compared to a Json file which is built up by brackets and namespaces.

When researching what interfaces, wrappers and other software and examples where available, Json came out as the format which had the most documentation on many levels. This was an important discovery, since more documentation and examples can reduce development cost. With its readability discussed earlier, Json was found the most desirable format to meet the requirements for a file based system.

3.2 Post-processing – Output file

The review reviled that there were fewer file formats to evaluate for the output result database. On the other hand, these formats have other properties that needs to be evaluated, than seen in the input file evaluation. It is desired to find an output format that have a C# interface. This would make it easier for Focus Konstruksjon to read the result database, without having to run any instances of the C++ written OOCfem. Another benefit of implementing a C# wrapper is that the threshold for maintaining and further develop an output file parser for Focus Software would be smaller.

As it is for the input format, the technical data structure parts are similar. The advantages and disadvantages of the formats will be investigated from a business case point of view, with the functional and the non-functional requirements seen in chapter 2. In this chapter a self-defined format will not be suggested. The reason is that the requirements of a third-party visualization program and that the post-process phase files are more complex than the pre-process phase files. The performance requirement would probably lead to that some parts being binary for the output file.

The sub-chapter will look at four different formats. These are chosen from a general research, for being investigated further as output files. Research shows that these are broadly used formats, and represents formats that are open sourced with well documentation, and formats that are developed by large commercial software companies.

Some of the file formats researched and evaluated will be presented in the appendix chapter D. Although, these formats are still presented in the table seen in chapter 3.3.

3.2.1 List of File Formats Found

HDF5 – Hierarchical Data Format

Hierarchical Data Format 5 (HDF5) is a file format designed to organize and store large amount of data [27]. The software library that is developed with the format is well-defined and well-documented. It has high-level API's for C, C++, Fortran 90 and Java. With a well-defined library. Research shows that HDF5 have been broadly used and have third-party connections, such as JSON.

HDF5 is supported by a third-party visualization software, such as Paraview. However, it is not possible to open a HDF5 directly in Paraview. HDF5 which is easily converted to other formats, uses a conversion to a VTK file format, which makes it possible to read HDF5 files in Paraview.

Figure 3.5: HDF5 Group

There are no high-level interfaces from the HDF5 group that supports C#, that are up to date. However, there was an effort supported by Agilent and Boeing to make a .NET wrapper called HDF5DotNet. The wrapper is a C++ subset of the API for native C++ code, that integrates C++ managed code for making a .Net framework assembly [28]. It became unsupported and unmaintained in 2012 and now the HDF Group encourage people to use P/Invoke instead. This a wrapper based on the same concepts as for the HDF5DotNet wrapper, and they do often pinpoint that it is not meant to be a high-level interface for the .NET framework. Even if the HDF Group claims that they support and maintain the P/Invoke wrapper, comments on their Github readme page does not substantiate that statement: “we blame all the bugs on Microsoft or The HDF Group” it says [29].

VTK

Visualization Toolkit (VTK) is an open-sources software created to handle 3D computer graphics, image processing and visualizations. VTK uses a wide variety of visualization algorithms to present scalar, vector, tensor, and texture data [30]. The format has high-level API’s for C++ and Python. VTK is tightly connected to a third-party visualization program called Paraview, which can present post-simulation data from structural analysis. VTK also have GUI toolkits to visualize its own formats, however Paraview is one of the preferred software for result database visualization after simulations.

VTK offers a format that is primary based on grids, and do also offer image data and poly data as examples. The vtkStructuredGrid and vtkUnstructuredGrid are the two data types that are interesting in this research, especially the unstructured grid since it often represents finite element data structures.

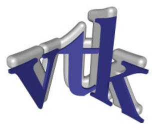


Figure 3.6: VTK - Visualization ToolKit

From research, it is found that VTK has an extensive user mass that gladly share experiences in a forum exclusively for VTK. Together with a well-defined and well-documented format,this gives a benefit in using and implement the VTK format.

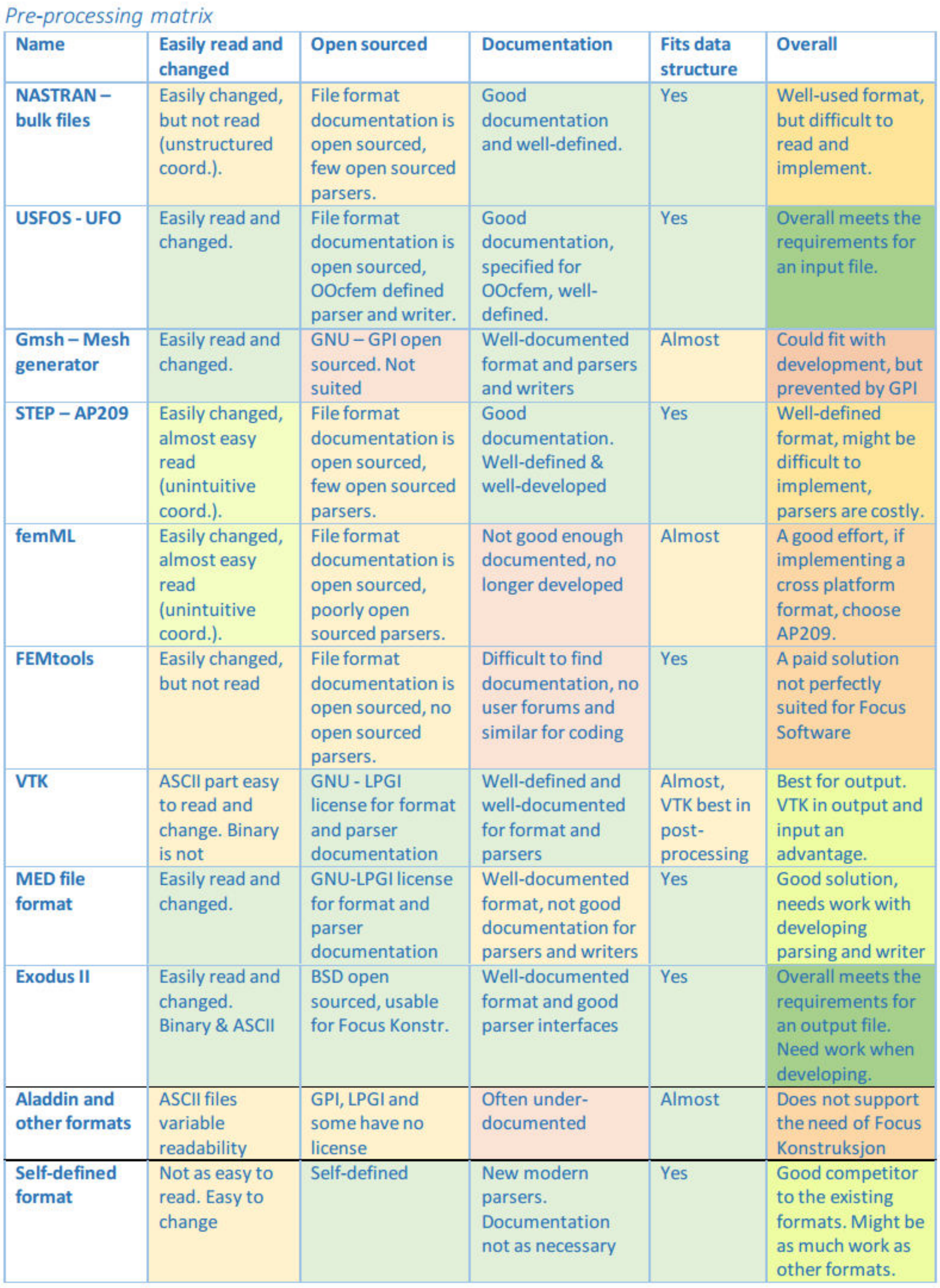
There is a .Net wrapper library for VTK called Activiz, which should contain .Net wrappers for all VTK objects, including Unstructured Grid [31]. The wrapper uses the same open source license as VTK, seen in chapter 3.1.1. It is well-defined and has code examples for different VTK formats and routines. In a file based system in Focus Konstruksjon a VTK file wrapper like this, would make it possible with an C# written parser to read transfer data from an external data structure. Other wrappers have been made earlier, but this wrapper is the one recommended from VTK.

There are two versions of the VTK format, the older Legacy and the newer XML format. The Legacy format uses an ASCII header with the possibility to store rest of the data as either ASCII or Binary. The newer XML format, encloses the different data sections in XML tags, where the data also here can be stored as ASCII or binary. There are also different extensions to these two VTK formats, when storing for example polygonal data (.vtp) or unstructured grids (.vtu)[32].

3.3 Matrix for Pre-and Post-Processing

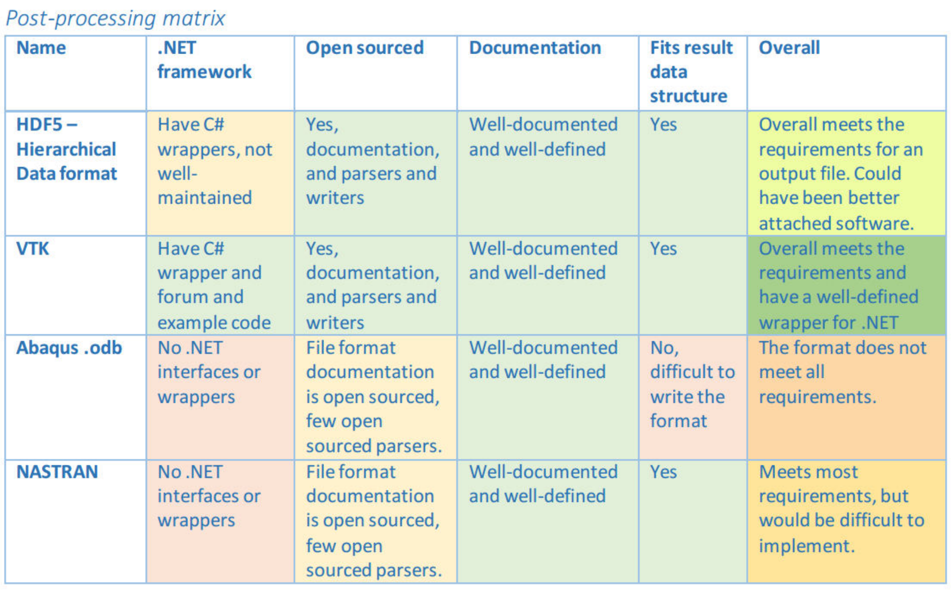
The columns of the matrixes are the essence of the requirements the file formats have been evaluated up against. As mentioned earlier in the chapter, some of the formats evaluated in this thesis can be found in appendix chaper D.The colors in the table represents how well the formats cover the requirements in the four columns. From that that well (red), and then different shadings, through yellow, up to very well (which is green).

Table 3.1: Pre-processing Table



Pre-processing matrix

Name	Easily read and changed	Open sourced	Documentation	Fits data structure	Overall
NASTRAN – bulk files	Easily changed, but not read (unstructured coord.).	File format documentation is open sourced, few open sourced parsers.	Good documentation and well-defined.	Yes	Well-used format, but difficult to read and implement.
USFOS - UFO	Easily read and changed.	File format documentation is open sourced, OOcfem defined parser and writer.	Good documentation, specified for OOcfem, well-defined.	Yes	Overall meets the requirements for an input file.
Gmsh – Mesh generator	Easily read and changed.	GNU – GPI open sourced. Not suited	Well-documented format and parsers and writers	Almost	Could fit with development, but prevented by GPI
STEP – AP209	Easily changed, almost easy read (unintuitive coord.).	File format documentation is open sourced, few open sourced parsers.	Good documentation. Well-defined & well-developed	Yes	Well-defined format, might be difficult to implement, parsers are costly.
femML	Easily changed, almost easy read (unintuitive coord.).	File format documentation is open sourced, poorly open sourced parsers.	Not good enough documented, no longer developed	Almost	A good effort, if implementing a cross platform format, choose AP209.
FEMtools	Easily changed, but not read	File format documentation is open sourced, no open sourced parsers.	Difficult to find documentation, no user forums and similar for coding	Yes	A paid solution not perfectly suited for Focus Software
VTK	ASCII part easy to read and change. Binary is not	GNU - LPGI license for format and parser documentation	Well-defined and well-documented for format and parsers	Almost, VTK best in post-processing	Best for output. VTK in output and input an advantage.
MED file format	Easily read and changed.	GNU-LPGI license for format and parser documentation	Well-documented format, not good documentation for parsers and writers	Yes	Good solution, needs work with developing parsing and writer
Exodus II	Easily read and changed. Binary & ASCII	BSD open sourced, usable for Focus Konstr.	Well-documented format and good parser interfaces	Yes	Overall meets the requirements for an output file. Need work when developing.
Aladdin and other formats	ASCII files variable readability	GPI, LPGI and some have no license	Often under-documented	Almost	Does not support the need of Focus Konstruksjon
Self-defined format	Not as easy to read. Easy to change	Self-defined	New modern parsers. Documentation not as necessary	Yes	Good competitor to the existing formats. Might be as much work as other formats.



Post-processing matrix

Name	.NET framework	Open sourced	Documentation	Fits result data structure	Overall
HDF5 – Hierarchical Data format	Have C# wrappers, not well-maintained	Yes, documentation, and parsers and writers	Well-documented and well-defined	Yes	Overall meets the requirements for an output file. Could have been better attached software.
VTK	Have C# wrapper and forum and example code	Yes, documentation, and parsers and writers	Well-documented and well-defined	Yes	Overall meets the requirements and have a well-defined wrapper for .NET
Abaqus .odb	No .NET interfaces or wrappers	File format documentation is open sourced, few open sourced parsers.	Well-documented and well-defined	No, difficult to write the format	The format does not meet all requirements.
NASTRAN	No .NET interfaces or wrappers	File format documentation is open sourced, few open sourced parsers.	Well-documented and well-defined	Yes	Meets most requirements, but would be difficult to implement.

Table 3.2: Post-processing Table

3.4 Final evaluation

The aim of this thesis is to end up with one input file format and one output format. For the input file the result of the evaluation showed that two formats, Usfos and Exodus II were the two existing file formats that was best suited. The thesis therefore shows that both the developer of OOCfem and the thesis research agrees that the Usfos format is a satisfactory way to represent the pre-processing structural data for OOCfem. Of these two it is known that OOCfem already have a parser and a writer developed specialized for Usfos. This also implies that the developer of OOCfem has knowledge about the format. All these factors make the Usfos format more benedictional than the Exodus II format. For Focus Konstruksjon, which are going to write the data through a pre-processing writer, both formats would be sufficient. Since a writer in C# will have to be developed, because there are no format specific software for these formats.

The self-defined format should not be evaluated up against the Usfos format on the same basis as other existing file formats. Since the benefits of these two formats are covering different elements of the requirements. Especially in the implementation phase. The self-defined format is expected to have a shorter implementation time. The format is however not as readable and does not build on a model structure as Usfos. As wekk as OOCfem will still be dependent on the API, which wont make the tightly coupled software any looser. It represents the data structures of the method calls to OOCfem's API. Usfos again is expected to take longer time to implement, even with the parser and writer developed, because of the different abstraction levels. The parser and writer are not finished and fully operational if new data elements are added to the simulation by Focus Konstruksjon, such as shell elements. It is not known how much new elements will affect the implementation of an Usfos format. Knowledge is gained by evaluating the formats even further when implementing prototypes of the two formats in chapter 4.

VTK and HDF5 resulted in the best suitable formats for the result database output file. The differences between the two formats are few, but there are some. It is desired that the output format is read by Focus Konstruksjon, without having to run any instances

OOCfem. Therefore, c# .NET interfaces or wrappers that support an output format will be beneficial. The evaluation shows that the VTK c# wrapper is superior to the HDF5's wrapper in terms of further development, documentation and a user mass that contribute to improve the wrapper. Again, some formats have already some implementations in OOCfem. There is a VTK writer, for the old legacy format, developed in OOCfem. From a business point of view, all these factors lead to less resources used when implementing VTK later, than HDF5. A requirement is that the output file should be read by a third-party visualization software. VTK has a tight connection to Paraview [33], whereas HDF5 does not. Over all VTK meets the requirements better than HDF5 do.

Chapter 4

Prototyping and Choosing File Formats

To have a better understanding of the issues and the complexity of implementing a file based system in Focus Konstruksjon, efforts were made in familiarizing with the code base of both software programs, in parallel with researching formats. This parallel work made it easier to see how findings from the file format research in chapter 3, applied to the internal data structures of Focus Konstruksjon and OOCfem.

After recommending what file formats Focus Software should investigate further, prototyping of the formats were conducted. After investigating the prototypes and their advantages and disadvantages, a decision is made for one input file and one output file for Focus Software.

4.1 Get to know Focus Konstruksjon and OOCfem

Familiarizing with a structural analysis software, demands effort and time. The abstraction levels described earlier made it easier to separate the code that would be influenced by a new file based solution, from the rest of the program. Despite the reduction, it resulted in a code base with a five-digit number of code lines. Another challenge was that the general initial competence of the thesis writer was in the object-oriented programming language Java. Focus Konstruksjon is written in the object-oriented language C#, which was found not that complicated to read. Compared to C++ language used in OOCfem, which has syntax rules that differs from C# and Java.

In this thesis the majority of the time was spent on code familiarizing and researching the call stack of different methods. First, how Focus Konstruksjon called the API of OOCfem was researched. The different elements in a FEM model and how the data transfer was between Focus Konstruksjon and OOCfem. Then going into OOCfem, analysing the process of reading FEM element data into internal data structures and then looking at the representation of the results database. Finally, how the simulation result database is transferred from the internal data structures in OOCfem to Focus Konstruksjon's internal data structures. The goal of these examinations of the data flows, was to get an essential understanding of what challenges would occur in a new file based system.

These code examinations were done in the earlier mentioned development environment, Visual Studio. Because the environment provides important tools such as call hierarchical searches, where it is possible to see how methods are called and how data flows. Peak of definitions to observe the internal data structure, was also an important tool. Together with a debug function that can recreate the call stack, of for example what is happening during a linear analysis. These convenient functions are a result of a complex environment which is strenuous to learn. This lead to a time consuming self-learning period. However, the knowledge acquired in this learning period will be beneficial later in the development process, later described in chapter 6. Visual Studio has also a lot of preferences that can be set. Therefore, in the beginning time was used to compile, build and set preferences in the different projects. As an example, it took two persons almost a day to resolve a simple build error issue due to some wrong checked preferences.

That is often what have happened when implementing prototypes as well. The actual time coding is minor compared to the time checking if different preferences are set correctly to build the code. Often accompanied with reading different forums to resolve the problem.

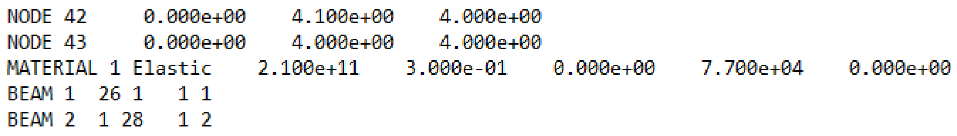
4.2 USFOS - prototyping

This Usfos implementation sub-chapters, will not focus too much on the actual implementation, rather the knowledge acquired and the results.

One of the reasons to evaluate USFOS further is that OOCfem already have an implementation of a parser and a writer for this format. A large part of the process of implementing a prototype was therefore to understand the Usfos parser and writer (from now called the Usfos-class). Looking at the abstraction levels, it was chosen that the input file routines should be written in C++, utilizing the benefit of the already implemented Usfos-class. The Usfos format is as known, well-documented. Therefore, the Usfos-class handles the conversion of the internal data structure to the file format well, and the other way around for a simple beam model. The Usfos-class is defined for the basic construction FEM-models, except composites and other more special constructions. What is not defined in the format is for example an analysis specific header, which could instruct what analysis should be run in OOCfem. To solve this issue, a corresponding header file should be stored together with the Usfos file, to not over complicate the format. This leads to that one header file and one Usfos file should be written for each analysis.

The prototype shows that it is possible to run a linear analysis of a relatively simple beam construction from Focus Konstruksjon, and store it as a Usfos file. With all FEM elements represented correctly according to the Usfos file format documentation.

Further, the same Usfos file was read during a Focus Konstruksjon linear analysis. The data was correctly read into the internal OOCfem structure. However, it was discovered that the data structure used for Focus Konstruksjon simulations in OOCfem and the data structure created by the Usfos-class were different. Where for example the distinction of node loads were described as CFNodeLoadData for Focus Konstruksjon and NodeLoadData for OOCfem. This shows the impotance of developing and writing software that represents the right structure and abstraction level. Nevertheless, the Usfos prototyping showed that it is possible to both write and parse Focus Konstruksjon's model data, with OOCfem's Usfos-class.



```
NODE 42     0.000e+00    4.100e+00    4.000e+00
NODE 43     0.000e+00    4.000e+00    4.000e+00
MATERIAL 1 Elastic    2.100e+11    3.000e-01    0.000e+00    7.700e+04    0.000e+00
BEAM 1  26 1   1 1
BEAM 2  1 28   1 2
```

Figure 4.1: Illustrating a selection of a Usfos file

4.3 JSON based self-defined format - prototyping

The self-defined format approach was, as described in chapter 3, an attempt to find the easiest and less time consuming way to transfer data from Focus Konstruksjon to OOCfem. Since OOCfem already have an API towards Focus Konstruksjon the purpose was to find a format that was well-known, well-defined and well-documented that could utilize that benefit. Json was chosen as the preferred format.

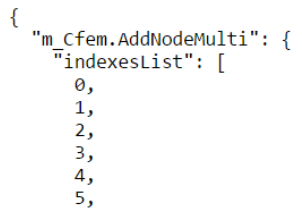
Focus Konstruksjon is developed in C# and the json format have different developed parsers and writers for the format that supports C#. The interface that was used the most on the .NET framework was Newtonsoft's .NET json plugin. In Visual Studio the integrated package downloader NuGet was used to add the interfaces in the right directories. However, as experienced many times before with Visual Studio, some configurations were wrong. That meant the time used programming was minimalistic compared to the time used on configurated Visual Studio.

For a faster and easier implementation, a dynamic Expando-Object was used. That is an object that expands when adding new declarations to it. The purpose of the self-defined file was to store the same data that the method called the OOCfem API with.

Therefore, the Expando-Object where given declarations of the same data structure as the API-method was called with. The prototype was developed to store the method, adding nodes, in Focus Konstruksjon. The node-object includes a list of indexes, a list of coordinates, a list of boundary conditions and a list for translations. Further a dictionary, defined by a string and an object, was used to make a data structure where the method name(string) was mapped to the corresponding method data (object). This was then serialized and stored to a json file.

The json file is then needed to be read by the C++ written OOCfem. Json is not as much used for C++ compared to C#. Still, there are interfaces available for parsing and writing json in C++. An API-class in OOCfem parses the json data and reads the string content. The string name in the dictionary has the same name as the method calls in OOCfem's API. Therefore, a search through method names in the API, will then find the right API method to call with the data stored in the object.

The benefit of implementing this type of a file system is that, compared to the Usfos implementation, it will reduce development cost. Choosing a self-defined format also makes it easier for maintenance and further development later on. This is because the self-defined file system is less complex than Usfos, and have more available software associated resources.



```
{
  "m_Cfem.AddNodeMulti": {
    "indexesList": [
      0,
      1,
      2,
      3,
      4,
      5,
```

Figure 4.2: Illustrating a selection of a self-defined file

4.4 Discussion and Choosing file formats

The evaluation of the file formats for an input file showed that most file formats covered the needs of representation of Focus Konstruksjons data. This lead to an evaluation focusing on the functional and non-functional requirements as a business case evaluation. The technical evaluation was done in the prototype implementation after selecting two relevant file formats.

As mentioned in the introduction, this thesis is not built up as a typical science paper. Where a hypothesis is presented and either backed up or disproven by theory. While looking at the file formats available for a new file based system, the typical approach with theories are not followed. That is because of fast-developing technologies within computer science and formats in general. The results of this is that an impression of how the market, technologies and file format functions often created over time by researching the field of relevant information and sources for a file based system. This causes difficulties pointing out an exact article or paper that backs up this impression. Gathering all this information to create that impression is the research of this thesis. It is believed that the thesis answers the problems described in chapter 1 with this approach. A good example of impressions created during research, is when reading and gathering information about a system or software in forums. Where different user share their knowledge and experience. The information from that kind of research is highly valuable for business case dependencies, such as easy implementations and performance. However, this information is difficult to back up with references, which results in this thesis own research on that related field.

When evaluating the formats from a business case perspective, the thesis had to find a careful balance. Of choosing requirements that separated the formats regarding quality and usefulness, and at the same time not having requirements excluding important formats. It is believed that this balance is found in the thesis, and the results of the file format evaluation showed that the requirements created a clear and informative separation of the formats.

Selecting input formats were challenging in terms of evaluating how much resources had to be put into implementing it. One format could have a data structure that matched the internal structure of OOCfem better than other formats. Even if the documentation on how to implement parsers and writers were under-documented and would lead to a time-consuming development process. It is believed that the thesis emphasis the value of the previously implementation of the Usfos-class in OOCfem the right way. By analysing the development costs differences. All file formats demand a certain learning period for a developer, to get familiar with the structures and the technical challenges. The thesis believes that this learning period, for a file format that already has some implementations in OOCfem, will be much shorter compared to others. As well as there is a valuable knowledge of the formats acquired by the developer of OOCfem. This could be very helpful for a full implementation of the Usfos format, in addition to lower development cost.

As expected Usfos was successfully implemented as an input prototype, with the already existing parser. However, the model used when prototyping was a simple beam construction model. The Usfos-class supports the most common FEM file element types. In the future, more complex geometric models will be written from Focus Konstruksjon, e.g. composites. The Usfos format supports the composit elements, however how the cross sections of the element should be written is not well described in the documentation. Prototyping showed that changes have to be done to both the format and the parser.

The self-defined format prototype implementation showed that it is different approach than for the existing formats. Instead of having a model data structure it is a method data structure. The format was found slightly unstructured and not as readable as the Usfos format. It will also be difficult to change the format, because some of the same data might be several places stored in the file. Changing the data one place in the file, might have dependencies in other places in the file, which could result in errors while parsing the file. An improvement of the file could be that lists that represents the same data, should only be stored once. With that change the file will be smaller and easier to modify. That being said, the benefit of the low implementing cost of the self-defined format might then disappear. The discussion on what format that should be chosen, should be focused on the balance between a format that is a representation of a model which requires a thorough implementation. Resulting in easy read and easy changed format. Or a format that is a representation of the method data already existing in the today's API solution, that does not require a complex implementation, but have lower readability.

The thesis experienced that considering abstraction levels when implementing the input format as important. Usfos provides a much deeper implementation of transferring data from a format into the data structures of OOCfem. The self defined format gave a solution using the same data as today's system, which would still use the API. Since the API only is created to serve Focus Konstruksjon, implementing the Usfos format on a deeper abstraction level would make OOCfem having more control of the input data and providing a more general interface. In the decision of choosing the input file format, the knowledge of OOCfem's developer should be seen as an important factor. Since, further development and maintenance are believed as important for Focus Software not only the next year, but over a horizon of decades to come.

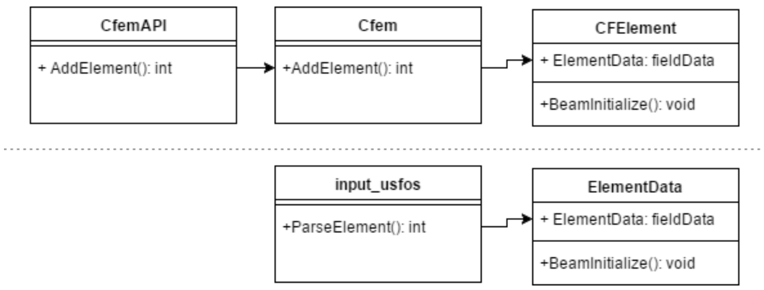


Figure 4.3: Illustrating the abstraction levels using API or the file based system

In figure 4.3 we see that the API based system, the top one, first needs an API class receiving the data. Second, it has a general Cfem class that converts the data that again generates the CFElement class for the elements in the geometric model. This is an an example for adding elements in OOCfem. Even the Element class is written exclusively for Focus Konstruksjon. Implementing the Usfos format will result in the lower system shown in the figure. Where the format is written directly into the structure and written to ElementData which is the original Element class in OOCfem. This is the same difference described with node loads earlier. As mentioned this gives better control of the data, with fewer classes and fewer conversions. Together with creating a more genereal solution for input data. Filling the requirements for this project and facilitate a more general OOCfem, the choice became to implement the Usfos format. Espcially after taking needs of both developers of Focus Konstruksjon and OOCfem into account.

An output format has not undergone a prototype implementation as the chosen input formats have. It was more important to evaluate the implementations of an input file than an output file, because of the number of different options faced regarding the input file. The VTK format was chosen as the output file, because it was the most suitable format found. It had the best parsing and writing software available, for the .NET framework, and already had a basic c++ writer for the VTK legacy format implemented in OOCfem. It could be discussed that not enough file formats were evaluated for the output format. However, when researching the field, some formats stood out, that is why only four formats were evaluated. Performance is a requirement seen in chapter 2. During research it was not found any especial differences regarding performance of the evaluated formats. Therefore, not given much room in this thesis. When taking all requirements into account, it is not believed that the thesis could find any post-processing phase file formats that have the same advantages for Focus Konstruksjon as VTK.

Considering that OOCfem already had implementations for parsing and writing the Usfos format and writer for the VTK format, the result of choosing those formats might be expected. However, the thesis shows that it could not be certain that these formats are the preferred once for Focus Konstruksjon. The Usfos format would not have been the clear chosen existing input format, if it was not for the already implemented parser and writer. Where Exodus II could have been the option. The thesis showed anyway that the Usfos format was the better choice. The disadvantage of choosing software that is already implemented is that the developer, of the file based system, will not develop the system from the beginning. This might result in much time understanding the implemented parser and writers. In addition, not writing the whole project using the same coding convention. However, it is believed that the disadvantage described is minor to the advantage of having OOCfem specialized software available.

F# was recommended from Focus Software to be researched as a possible programming language for implementing a parser in Focus Konstruksjon. From the results presented in the thesis, it is known that the output format has a C# wrapper available. The thesis project will therefore prioritized C# as the language to use on Focus Konstruksjon side of the project.

Chapter 5

Test Driven Development

Choosing the file formats for the file base system, 4.4, enables the development and implementation of the system. Before fully focusing on development and implementation, which is presented in chapter 6, this chapter will describe one approach to drive the development process, Test Driven Development.

5.1 Format of the process

This thesis used the Test Driven Development (TDD) process [34] to drive some of the development. The process uses a short development cycle:

1. Write an initially failing unit test. The test should represent a desired improvement or functionality. For example, a not written sum-method, could have a test which checks if one plus one returns two. Because the sum-method does not have any functionality yet, the test will fail.

2. Then produce minimum code that is necessary to pass the test. In our sum-method example, the minimum code to pass the test would be for the method to only return the integer two, without doing any calculations.

3. Refactor the produced code, to the acceptable code conventions used. Which in

the sum-method case, is not necessary.

After completing the cycle the developer has created a unit test. To further follow the TDD process the developer starts at the beginning of the cycle (step 1). Referring to the sum-method example, the next step could be to write a unit test checking that one plus two should equal three, which at the moment fails (step 1). Next step in the cycle would be to produce the minimum code to pass the test, which would be to allow the method to take two parameters, and return the sum of these two, in order to avoid failing the first test (step 2). Make sure all tests will pass after refactoring (step 3)

Arrange, Act and Assert

The TDD process is based on unit tests, with the three step cycle to drive the process. The same number steps are used for creating a unit test. It is known as the Arrange, Act, Assert-pattern, or shortened to the AAA-pattern [35]. The intention of these steps is to ease the understanding and readability of the test, which again leads to better maintaining. The pattern is divided into these functional sections[36]:

1. Arrange all necessary preconditions and inputs
    - For the sum-method example used earlier, this e.g. implies to initialize the sum-method class.
2. Act on the object or method under a test
    - Further, during the act section the sum-method from the sum-class is called to act the sum of two numbers.
3. Assert that the expected result has occurred
    - Finally, the result of an act is asserted, where the expected result should be equal to the actual result. If the sum is expected to be three, the result is checked if it equals to three or not.

5.2 Benefits of TDD

The main goal of writing unit tests in TDD is to force the focus of the developers from initially focusing on the actual code, over to first focusing on the requirements and specifications of the software. Requirements and specifications are necessary for the developer to understand, to be able to know what a unit test actually should test. Even if there is expected an extended time developing utilizing the TDD process, professional programmers achieved a 40-50% improvement in code quality [37]. The improvement is gained by focusing on the requirements and the specifications first. For complex projects with a longer project time horizon, the time used to develop, maintain, and support a software would decrease using Test Driven Development because of the time saved having good code quality and fewer errors [38]. Therefore, by originally concentrating on what a unit test should test, development cost could beneficially end up being reduced.

Code error-security is also one of the main benefits of TDD. The tests will provide the security that if code is changed, it will warn the developer that functionality of the code is damaged. If a developer is extending the sum-method example developed by using TDD, he or she could change the method having the security of not breaking functionality. Because all specifications and requirements has been taken into account when the unit tests were written. This benefit of TDD becomes especially important if the sum-method is part of a complex software program, and other parts of the software may break due to the changes in the small simple sum-method. This advantage helps developers sleep at night and confirm that what they have developed actually work.

Another important benefit is efficient debugging. If any changes made causes an error, the tests will provide information about where and what the error is. This becomes highly important for maintainability and for code projects where several developers are contributing to the same project.

The TDD process is a rather simple method to follow, which is one of the reasons it is a powerful tool. From the beginning of a project, TDD encourages developers to divide the challenges and tasks into smaller parts, where the developer must fully understand the purpose and the goal of each task. This can be tightly connected to a brake-down structure which is an essential part of agile development. Each unit of the project, can be built and tested one by one, and will improve the development, testing and maintenance. The more complex a project is, and the longer a project's time horizon is, there will be developers that come and go, someone might get sick, fired, or finds new jobs. To make sure that the knowledge of the developers is represented in the code, TDD developed tests will show what challenges, requirements, and specifications is thought and taken care of by every developer in the project.

5.3 Limitations when using TDD

The most apparent disadvantage of using the TDD process is the time used developing tests which drives the development. This could for example imply that for a class needed in the project with 20 code lines, could end up having a test class of for example 100-150 code lines. To test all necessary features. Extended time use leads to cost increase. Therefore, the interesting question is; when is it break-even and when is it favourable to use TDD?

The thesis did not find many companies using Test Driven Development in professional development. The research shows the opposite. This is probably because it is deadlines that run companies and not the benefits of satisfying code quality. With deadline-driven projects, projects becomes short termed. There are few arguments found arguing that TDD would benefit a short termed project process [39]. However, there are many arguments for long-termed project. The disadvantage of TDD is therefore that projects often has too short deadlines to fully benefit TDD. This is especially difficult to see from a manager point of view. The break-even is therefore difficult to find. However, the research shows that TDD should be more beneficial than companies seems to understand.

As mentioned above, using unit tests will lead to more coding. All code must be maintained, as well as the test code. Therefore, maintenance cost will increase with a unit test code base. The size of the test code base will also affect the cost of changing requirements or the architecture of a software. Which could be a risk for a project owner if it is expected to be changes to the requirements in the future.

It is not often that projects starts with writing code from scratch, and projects often have developers organized in teams, which results in complex code bases. If the code base is not testable, the code depending on the previous code might be difficult to develop further as TDD, because of dependencies in the complex code base. Encounter a complex non-testable code base is therefore a limitation to begin TDD development. To be able to write testable code, focusing on the code quality is important, which again will decrease the limitations of using TDD. The clean code principle and the SOLID principle is know as a good way to describe testable code. This is further described in chapter 6.6

From the code delivered for this thesis we see that for both the Usfos writer and the VTK parser there are unit and integration tests. The code is testable, because of loose couplings and it is following the single responsibility principle in clean code. However, since the writer and parser was written as prototypes in the beginning, with a code driven development approach (see chapter 6.4). There a mostly integration tests developed during this thesis project, to test the already implemented functionalists. Whereas for the OOCfem specific classes in Focus Konstruksjon, there is only integration tests because the code is less testable.

5.4 Integration tests, system tests, and acceptance tests

There are more ways to test a software then writing unit tests. In the beginning the we see that the developer follows the AAA-pattern to develop units or components of a software, through unit tests. From the figure 5.1 we see that the next step of building tests for the code base is to develop integration tests. These tests are developed to verify how different software modules works together. Integration tests seeks to find out how these methods are integrated into the software. The main purpose of integration tests is to search for defects in interfaces and interactions between modules or software components.

An example from this thesis is when reading an input and output file, an integration test will be testing files where the input and output is known. It is for example known that node one from the file should have certain coordinates. Since the different modules of the software should produce such results for the coordinates, an integration test can be written for this exact coordinate-case. If a developer changes the code later and breaks the interface and interaction between the modules, the integration tests will then reveal what part of the code broke.

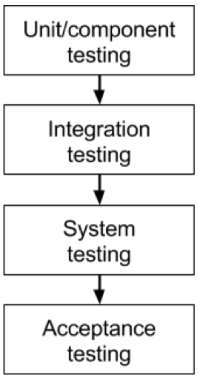


Figure 5.1: Overview of the test structure

System testing is focused on how the product/program meets the requirements. The tests are based on a complete integrated system. The requirements focused on in system tests are the functional requirements. Whereas for this thesis, a system test could check if the file structure is readable or that it can be read by an external visualization program.

Acceptance testing is tests that verify that a product/program is working as expected within the environment is it running on. It is seen as the non-functional requirements testing. Where performance, stability and reliability are some important factors. If Focus Software are introducing a file based system to their customers they need to trust the product, before they completely change the data transferring system in their software.

Providing tests to all of the four stages shown in figure 5.1 will provide confidence to the stakeholders that the product can be trusted. The same trust will not be achieved with only running acceptance testing. Because the tests are dependent of each other. The challenge in a deadline-driven marked is to not jump directly to system and acceptance testing. This can result in testing at the end of a sprint or a long project, which often becomes more time consuming then expected.

Chapter 6

Software Development

This chapter will describe the different stages of software development for the file based system. The stages described in this chapter is important for gaining the necessary overview of the project development and what decisions that have been made during the implementation. As this thesis is focusing on delivering a new software system for Focus Konstruksjon, it requires the reader to obtain most of the information about this thesis software project through the code. As mentioned earlier this is a backend project which makes it difficult to illustrate the development of the project. Therefore, this chapter is substantial for understanding the parts of the development and implementation process that is difficult to identify in the code.

6.1 Planning

It was early decided to develop both Focus Konstruksjon and OOCfem with a looser coupling and a generalization of the data transfer, which resulted in implementing the Usfos and VTK format. Making both software programs more general in terms of data transfer, via the chosen file formats, required changes and development not only on the top-level. In OOCfem the API should not be used as the data transfer point, the data should be read and written in a different abstraction level. This gives the opportunity for OOCfem to be developed and maintained more independently, instead of directly being shaped after the needs of Focus Konstruksjon. The OOCfem wrapper in Focus Konstruksjon was already developed as a data transfer point towards OOCfem. Therefore, compared to adjustments of OOCfem's abstraction levels, there was not necessary to focus on the core of the software in the same way.

Looked at from the outside, Focus Konstruksjon is a software that works seamlessly together with an API, which is provided by OOCfem. When further investigating the code and implementing prototypes, chapter 4, it was clear that the API was created with the goal of serving Focus Konstruksjon, and not as a general API for other software. The decisions on how data should be transferred and the structure of it, were taken care of by the API, which resulted in an API evolving after the needs of Focus Konstruksjon. With the new file based system, these previous decisions had to be implemented again to get the same desired results as with the API. This sometimes lead to a situation where planning were time consuming. Since questions about how the data structure should be formed had to be answered, and changes to the structure that differed from todays solution had to be discussed.

However, with tight communication in weekly meetings and e-mail correspondences it was possible to develop with some of the agile principles, which provided a continuity of the thesis project. Where both OOCfem and Focus Konstruksjons developers were stakeholders in the project, and decisions on how to develop and implement the file based system was continuously discussed and changed. With these meetings and discussions it was easy to form a break down structure, which again lead to simplifying the tasks to be solve during development. Problems occurring and challenges met was then successfully to implement into the task list, after discussing it with the stakeholders.

6.1.1 Architecture

An overall Architecture pattern used for Focus Konstruksjon is the Model-View-Controller (MVC) [40]. Since the model is the geometric model, controllers are the only way it is possible to interact and request results from the model, such as building the model or choosing how to analyse it. The view is how to present and visualize the model and the results. The file based system is oriented in the model part of the architecture pattern, where the new system only interacts with the data in the model. There are no views involved in the system, and it is not possible for a user to interact and control the system.

Further, an architecture pattern can contain other sub-architecture patterns. The pipe and filter architecture pattern is representing the structure of the file based system. The pattern is also known as the pipeline pattern [41]. How data flows through a system (pipes), while between the pipes the structure of the data is transformed to fit the required structures in the software (filters), is the basis of the pattern. See figure 6.1

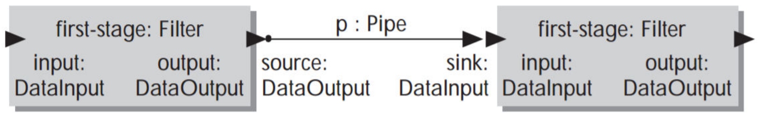


Figure 6.1: Overview of the basic steps of a pipe and filter pattern

From the figure we see how data is gives as input to the filter, then filtered and sent by using a pipe, to another filter. For example, this could be looked at on how the data is transformed in the file based system from a XML VTK file to double arrays in C#. It represents the same data, for example nodes with displacements, but is filtered in different ways to fit the software data structure. Figure 6.2 illustrates how the architecture of the new file based system is structured. Where the files represents the end of a data flow, before being used as input of another part of the system. The modules representing the "data structure" of the software are only simplifications of the core functionalists of the software, which is not relevant for the file based system.

All of the classes illustrated in the figure generates new classes and have a number of methods, which are not visible due to simplifications of the system. These classes and methods represents the most important basis of the file based system, and helps to get an overview of the core functionalities of the system. The first row of figure 6.2

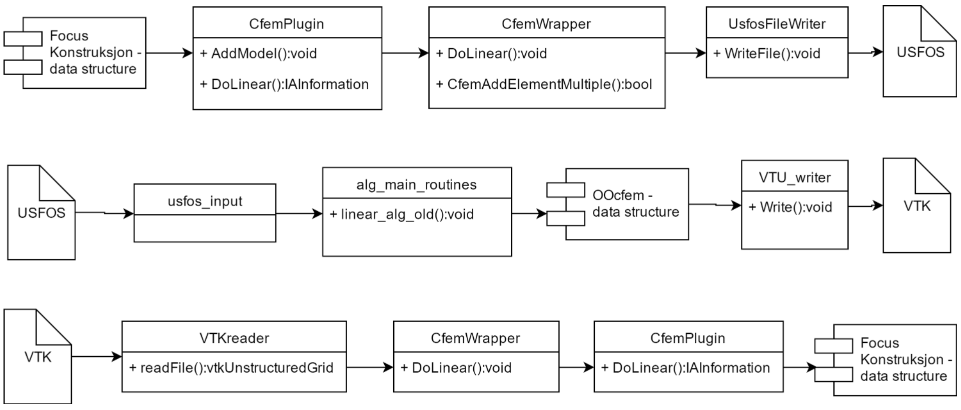


Figure 6.2: Illustration of how the architecture is following the pipe and filter principles

is illustration how the pre-processing phase is generating a output file. The Cfemwrapper wrappes the Focus Konstruksjons data structure and transforms the model data to a OOCfem required data structure. The figure is using FEM elements as the example data which is written to the Usfos file. The wrapper has a AddElementMulti method which transforms the Focus Konstruksjons data structure, to data that can be written to the Usfos output file. These Usfos file methods are all represented by the namespace WriteUsfosFormat. See figure 6.3 for a class overview. Which also illustrates the complexity of the system, and how figure 6.2 is simplified to only illustrate the top abstraction level.

Another example is the third row of the figure 6.2, where data from the VTK file is read by Focus Konstruksjon. The CfemWrapper uses the namespace ParseVTKFormat, when reading forces, stresses and displacement from the VTK file. This is a filter method transforming data from the file, which later is used by other parts of the software. An example of the class overview is seen in figure 6.4, which represents the DoLinear method seen in row three in figure 6.2

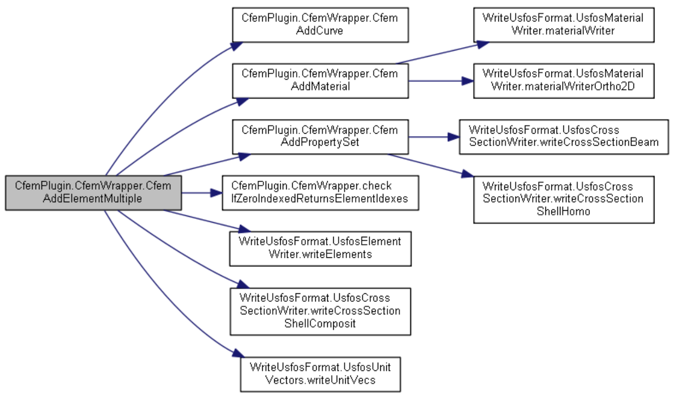

CfemPlugin.CfemWrapper.Cfem AddElementMultiple
CfemPlugin.CfemWrapper.Cfem AddCurve
CfemPlugin.CfemWrapper.Cfem AddMaterial
CfemPlugin.CfemWrapper.Cfem AddPropertySet
CfemPlugin.CfemWrapper.check IfZeroIndexedReturnsElementIdexes
WriteUsfosFormat.UsfosElement Writer.writeElements
WriteUsfosFormat.UsfosCross SectionWriter.writeCrossSection ShellComposit
WriteUsfosFormat.UsfosUnit Vectors.writeUnitVecs
WriteUsfosFormat.UsfosMaterial Writer.materialWriter
WriteUsfosFormat.UsfosMaterial Writer.materialWriterOrtho2D
WriteUsfosFormat.UsfosCross SectionWriter.writeCrossSectionBeam
WriteUsfosFormat.UsfosCross SectionWriter.writeCrossSection ShellHomo

Figure 6.3: Class overview of the AddElementMulti method in the CfemWrapper

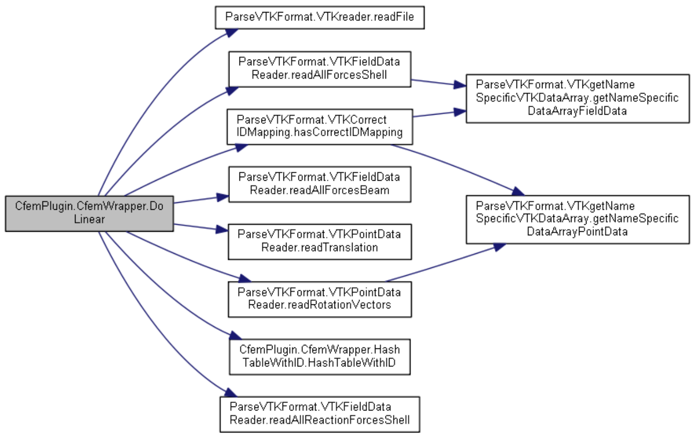

CfemPlugin.CfemWrapper.Do Linear
ParseVTKFormat.VTKreader.readFile
ParseVTKFormat.VTKFieldData Reader.readAllForcesShell
ParseVTKFormat.VTKCorrect IDMapping.hasCorrectIDMapping
ParseVTKFormat.VTKFieldData Reader.readAllForcesBeam
ParseVTKFormat.VTKPointData Reader.readTranslation
ParseVTKFormat.VTKPointData Reader.readRotationVectors
CfemPlugin.CfemWrapper.Hash TableWithID.HashTableWithID
ParseVTKFormat.VTKFieldData Reader.readAllReactionForcesShell
ParseVTKFormat.VTKgetName SpecificVTKDataArray.getNameSpecific DataArrayFieldData
ParseVTKFormat.VTKgetName SpecificVTKDataArray.getNameSpecific DataArrayPointData

Figure 6.4: Class overview of the DoLinear when parsing the results from the output file

6.2 Development

This sub-chapter will look at three difficulties encountered during the code development. For the reader this sub-chapter should describe what parts of the development process were time consuming and important, that did not result in actual coding.

Finding reasons to incorrect result data

When successfully writing a VTK file from OOCfem, the post-processing result database, a time consuming challenge could be to find reasons for errors occurring in the results. During the development, both API result data and file result data was read and the evaluated, for e.g. displacement, IDs and forces. It was then easier to compare if data were corresponding between the old system and the new file based system. When data did not correspond, the challenge was to examine what input data was not given to OOCfem through the USFOS format that was read by OOCfem through the API, or the other way around. Because OOCfem simulated the input data from both the Usfos file and the API with the same algorithms, and the old API-system was tested by Focus Software that it provided the correct data. It was certain that all result data from the API were correct, and the error had to be found the way data was written. Either the lack of data written from Focus Konstruksjon or data written incorrectly to VTK from OOCfem.

This often lead to a time-consuming debugging period, where function calls and function hierarchies had to be evaluated. After debugging, a meeting with the product owners of both Focus Konstruksjon and OOCfem to discuss the findings, often lead to a strategy of discovering what could cause the irregularity. As an example, the element direction of beam elements got set as a default in OOCfem, if the USFOS file did not specify it. The simulation completed with no errors, but did not give the expected results. This lead to a time consuming period of debugging and discussions, which revealed that OOCfem and Focus Konstruksjon runs with two different definitions of the directions of elements.

Simplifying data objects

When a method got to many input parameters, the method did not correspond to the clean code concept, seen in chapter 6.6. To make the method parameters more readable and understandable, a bean class was implemented. Inspired from the JavaBeans class [42]. This class only holds data, with no additional functionalists such as event-handling methods. Similar to an Object, other than that this class holds different types of objects. With this type of class the developer does not need to take into account that the data might have different data types, such as string and int. This simplifies and generalises the development process, and makes it easier to add data types to a method without having to change the method parameters. This have for example been used when writing Usfos elements in Focus Konstruksjon, called UsfosElement.cs.

Where to store files - Application Data (AppData)
During the development of the file based system the question of where to store input and output files became an issue. Earlier, VTK Legacy files had been written to the local AppData folder from OOCfem. However, Focus Konstruksjon had generated a application folder in local directory My Documents for the user. The question then was what type of abstraction level for directories was preferred for the file base system. Microsoft creates the AppData folder for every application, and the folder lives until it is deleted together with the application [43]. In that way the local AppData folder is suitable for the file based system. For the .NET platform this also provide a general solution for file paths that is not bounded by user restrictions and checks from Windows to allow writing files to the local storage. It was decided that the files should not be written at an abstraction level for the users to easily access the input and output files. The files should be written to the Application folder, AppData.

When a user opens the program, all application files are originated on the application folder (AppData) of Focus Konstuksjon 2017. In that folder, where OOCfem's API DLL's were stored earlier, the .exe file of OOCfem is now placed. The .exe application file is the file that takes in the input file and execute the analysis, and returns the result database in the file based system. The file based system is generating directories for input and output files, while these created directory paths are used by the .exe file. Therefore, the only thing the user needs to run the file based system is the application folder of Focus Konstruksjon 2017.

6.3 Implementation

The first implementation of the file based system was to implement code for conducting a linear analysis of one single beam element with just structural weight as loads. This is seen as a simple case where not much additional data is needed. However, the case provides important data structures for both the input and the output file, which has to be used in every case in a FEM simulation for more complex analysis later. Such as nodes, elements, forces, boundary conditions, cross sections and materials for the input file, and in the output result file; translation, rotation, stresses and moments.

Beginning with a simple beam analysis and getting correct results, was the first milestone during the implementation. Further, the focus of the thesis was then to fully complete all functionalities of a linear analysis with different input models (input files). Resulting in implementing the file based system for different load combinations (e.g. live load, wind and snow loads), shell elements with homogeneous properties, composite shells. and special cases of masses and couplings.

After fully completing the linear analysis, implementing the other structural analysis provided by Focus Konstruksjon would be much less time consuming. Because of the implementation of the linear analysis was generalized to easily fit the other analysis, which have to be developed to finalized the file based system.

Status on file formats – experience compared to expectations

In chapter 4.4 it was implement to develop the Usfos format as the pre-processing file format and for the post-processing phase, it was decided to implement the VTK format. During the development of the file based system the experiences of implementing and using the formats were compared to the expectations built up during research.

Usfos

The first necessary task for enabling usage of the Usfos format was to develop a writer for the format in C#, to be used on the Focus Konstruksjon side. Then further improve the parser implemented on the OOCfem side, to fit the rest of the data structures that had not been integrated into the parser yet. Implementing a Usfos format writer in C# was time consuming, although not very difficult because of a satisfactory documentation of the format. Together with the fact that the format is readable, which lead to a basic and understandable format to implement. The struggle with implementing the Usfos format was when the documentation did not describe how certain elements of the format were to be written. These situations were often time consuming because of the need of external help. Valuable information was also often difficult to achieve while debugging OOCfem, as a result of the size and complexity of the code base.

Parsing the Usfos file was not difficult for a simple beam construction, since functionality was already implemented in OOCfem for those kinds of constructions. Therefore, for these simulations the experience was that implementing the Usfos file did not face great challanges. However, when the documentation had to written to fit Focus Konstruksjons data structure, the process became time consuming. The extensions of the Usfos format, in terms of new components or additional fields in already existing components, can be seen in the appendix sub-chapter B.1. For an simple beam construction, a Usfos file can be found in appendix chapter B.

An analysis file for OOCfem, was to be developed together with the geometric model in the Usfos format. During the beginning of the implementation, an analysis file were developed. However, it was decided to integrate the analysis data into the Usfos format, to save development time. The fields of the analysis file were simple data types, often a word and a single numerical value. Writing the documentation for the analysis data was therefore not difficult. What was more struggling was to parse the file and assign the analysis values in OOCfem. For further development of the system it is important to know that there is functionality for writing and parsing an additional analysis file developed, but currently not in use.

VTK - .vtu

The thesis started the development process with developing a small prototype writing the VTK format, to see if the framework for VTK worked properly. It was experienced that working with the VTK framework in C++ was not straight forward, especially when using cMake to unpack the library. However, the framework for writing a VTK file and the well documented .NET wrapper, showed that using VTK for the C# side of the project, was the right choice. It was decided to use unstructured grid as the data structure for the VTK format. This again lead to the possibility of choosing the new

XML format for unstructured grids, which has a .VTU extension. This format could both be written in ASCII and binary, which was necessary for meeting the requirements of this thesis. The format has data organized under headings such as “BeamStressRes” and “NodeIDs”. With the XML format, especially with the ASCII version, this makes the file readable without any other additional explanation and documentation. The binary version of the format makes the file unreadable, but provides a more compact solution. Examples of the formats can be seen in the appendix chapter B.

The unstructured grid format provided a structure where the results could be referred to as data related to points (nodes in the FEM model) or cells (elements in the FEM model), which makes it possible to build a geometric model of the post-processing data. However, not all data could be written in that form for the result database. Elements could have nodes that overlap with other elements (one node representing more than one element), where one node could end up holding different values for different elements. Therefore, an array type called field array, which holds data unrelated to points or cells, was used to ensure that data was not overlapped or written incorrectly. Providing a functional VTK file with a geometric model and additional result data, which is important if the file should be read by Paraview.

Examples provided by the Activiz documentation, of how to use the VTK format was not as available as expected. However, the .NET wrapper, which was available with NuGet (NuGet is a free and open-source package manager designed for the Microsoft development platform (formerly known as NuPack) [44]), provided explanations of returning values and input parameters in Visual Studio while writing the code. With these explanations it was possible to understand the framework and what it was doing, while developing and debugging. The community using VTK and forums were expected to be of more help, but with the documentation available it did not lead to much additional time spent on understanding the format and its corresponding software.

In the beginning the Activiz framework gave the expected result. During the code driven prototyping the data that was parsed from the VTK file and the data from the OOCfem’s API seemed to correspond. However, when implementing integration tests, the tests reviled that the result data was not as correct as first believed from prototyping. Parsing data from the VTK arrays written in 32-bits from the C++ side and then using a wrapper that read the data in 64-bits, it seemed that precision decimals were added to the data during parsing of the file. This also was a adequate example of how tests makes controlling the code and the results much easier and secure. Researching documentation and examples did not provide any solution to this problem. Therefore, an e-mail requesting an explanation was sent to Kitware, who is developing Activiz. The response from Activiz was the following:

“ Data arrays of vtk mesh are stored in Float32 bits. So, I think that if you try to read it in 64bits, then some decimals can be added for more precision. I can suggest two solutions:
- Truncate your values after x decimals
- Try a 32 bits Activiz which shouldn't add extra decimals”

Since Focus Konstruksjon runs on a 64-bits system, the solution to this problem was to truncate the decimals that was added for precision while parsing the values. Since Focus Konstruksjon has a policy that numbers smaller than E-08 is to be evaluated as zero, and the decimals added during parsing was added after the 8th decimal, it turned out that the problem could be ignored. Because Focus Konstruksjon got the required precision.

Other than that, the Activiz framework worked well with the file based system. Since the pipe and filter architecture pattern made the solution focusing on a general solution for parsing values. The post-processing parsing filters makes it simple to convert data arrays into a VTK file and read the data. That implies that OOCfem now have a general way to write values from the C++ software into a VTK file, and receive the same values in a .NET C# solution. This provides a more general data transferring opportunity to the .NET platform, that OOCfem have not had before.

6.4 What has been driving the process?

Prototypes were developed for both the pre-processing and the post-processing format. The main goal of these prototypes were to ensure that the file formats chosen was working as expected and to see if the respective resources (such as wrappers) worked as described in the documentation. When developing the prototypes, the goal was to understand the wrappers and code base. Code was written to solve different tasks, for example writing nodes to the format from the node array. If it seemed to work, a next task was encountered. This is described as a code driven process. Where if the code gives the expected result, it means that the task is solved.

Later, when further developing small parts of the file based system. Such as e.g. small search methods, the TDD-process was used. It was believed that TDD should have been used more, which is elaborated in chapter 7. Therefore, during the prototyping first code was driving the process, and later tests were driving it. Which from a planning point of view preferable should be the other way around. Because TDD focuses much more on the requirements and the challenges of the code. Instead of just staring to write, as code driven development is known for.

During the implementation, it could be discussed, in a more humorous terms, that Jesus Driven Development has been used [45]. This is because sometimes during the development it was not sure how a problem should be solved. A solution was to just try without knowing if anything would crash. Further, after implementing the solution it was not possible to test properly that the change did not break any other code. The expression comes from hoping that everything will be working without checking it, also called Faith-driven development [46].

Another ting driving the development process was debugging. Using Visual Studios debugger to find out what was giving errors in the code. Even if the process could be time consuming and it could take days to resolve one single error. Knowledge acquired during the debugging process contributed to drive the development.

6.5 Documentation

Through this master thesis the goal has been to implement the file based system with a code quality that ensured minimum need of written documentation. The documentation is given in the form of understandable and readable code. Comments in the code are given where a more thorough explanation is needed. Variable, method and class names were chosen to give an understanding of the code and what the different parts of the code is handling, without having to comment and write documentation. Another way a developer can provide an understanding of the code is through writing tests. As mentioned in chapter 5, tests are often satisfying documentation of what challenges and requirements a developer has handled while developing a system. Over 100 tests are written, to ensure that the file based system is functioning according to the requirements after undergoing maintaining, further developing or changing of the code. This gives not only an security for future development, but also gives a satisfactory documentation of what the code does or is expected to do.

To easier understand the architecture and what dependencies are present in the new system, Doxygen is used. This is an automatic code documentation generator. The documentation can be seen in appendix chapter E.

Another form for documentation is the development of documentation for the Usfos input format. Were the OOCfem parser was improved to read new Usfos data elements, the documentation document was also expanded. These improved documentations can be found in appendix chapter B.1. In terms of the VTK XML format, the format is readable, and does not need any additional documentation then the VTU_writer class in OOCfem. If a more general understanding of the format is preferred, the format is well documented by VTK Kitware Inc [30]. Examples of both formats can be found in appendix chapter B

Documentation of the work progress could be found in the Commitment History in Team Service Visual studio (Focus Konstruksjon) and GitHub (OOCfem). Through the commitments it is possible to get a overall understanding of what have been changed and at what time.

6.6 Code Complexity

As mentioned in Chapter 5, writing testable code leads to an understandable, reusable, and robust code. The file based system was written with the TDD principle, where it was possible. In addition, the thought-set from the TDD process provided a requirement first attitude, which insured the code qualities mentioned were focused on during the development.

Focus Konstruksjon is written using the Object-Orientated paradigm, which should "reflect the real world" [47]. By that it is meant that code objects simulates objects from the real world and how they interact with each other. Coding in a real world reflection gives a solution with many classes that have different responsibilities, and not one big class that solves many challenges. This again encourage coding that reflects the Clean Code principle [48]. Where it says that a class is optimal if it only has one responsibility. Further, classes should be as small as possible and have methods that only solves one task. Preferable should the class contain maximum 30 methods. This is also described in the Single responsibility principle in SOLID, which is the "first five principles" for object oriented design [49]

The disadvantage of not having code written by the Clean Code principles was experienced during familiarizing with the code base of Focus Konstruksjon and OOCfem. With older classes containing more than 4000 code lines, with methods solving more than one task. These were difficult to understand and debug. This made the familiarizing process time consuming, especially when methods were coupled with a number of classes and properties. The couplings seemed to become tighter every time a new method, solving a challenge that the class initially was not made for, were added to the class, instead of creating a new class.

Another important factor in code quality is naming of classes, methods and variables.

The main goal is to prevent a developer using time reading documentation and debugging code to understand the simple elements of the code. Experience from the thesis showed that Focus Konstruksjons C# code have better clean code naming than OOCfems C++ code. It could be many reasons for that. One is that C++ lies closer to an older Fortran culture, where variable names should be shortened to prevent using more memory then necessary. Clean code stresses that the names could be long if they need to describe the functionality precisely, and that understandable longer names triumphs shorter names even if it saves data usage.

The file based system has followed the clean code principle during the development, as good as possible. However, research thorough forums showed that developing parsers and writers does not create the best foundation for developing clean code. Where a lot of data is read and written after certain rules, which creates a developing environment which are not always readable. Some places the code would seem redundant, because of the rules of the format in a file. On the other hand, the ground principles in clean code are followed to the best ability of the developer. For example, having developed many classes with a single responsibility, in addition to understandable variable, method and class names, makes the code more readable.

Chapter 7

Discussion

Before this thesis, OOCfem developer Bjørn Haugen has written code readjust to how Focus Konstruksjon defines models, geometry, loads, etc., by developing the API and functionalists to their needs. With the file based system, Focus Konstruksjon and OOCfem are more loosely coupled, providing a more generalized solution. This is an advantage for both software programs, in future development of Focus Konstruksjon. Another benefit is that OOCfem only gets one access point. This will generalize how OOCfem receives data, which again gives the owner more time to develop the software while not adapting to changes in other solutions.

It was beneficial for both software programs that the file based system was implemented now. As described in the introduction, the users are in need of the file based system. When the system is fully implemented, the users will save time and resources using Focus Konstruksjon, which will increase the value of the program. Focus Konstruksjon will also benefit from the file based system in terms of easier maintenance and debugging of their analysis part of the program. Considering that the API had grown to a size where it is complex and have functionality for many small and rarely used functions. Further developing the API would have lead to an even more tightly coupled system, which in the future would have made it difficult to develop and maintain the software. With the file based system, all managed code can now be removed, which reduces the disadvantages of running C++ on a .Net platform and gives the advantage of having less code to maintain. The input file and the output file also makes it easier to maintain and debug OOCfem. All in all, both software programs benefit from the new system.

As mentioned in chapter 6.4, the driving force behind the development process differed greatly. The TDD process is a time consuming process. However, it creates value to the code, especially after leaving a project and another developer takes over. During the development of this thesis, the TDD should have been utilized earlier. The prototyping of the parsing and writing parts were the once best suited for a test driven process, however the knowledge of the process was not acquired in time. However, the mind-set from TDD was important in later development, where developing code stone by stone was part of the main focus. Developing the simple requirements first, and not jumping straight to the difficult parts of a class or method.

The benefit of using Paraview has been described earlier in this thesis. During the development, Paraview was used a few times to check results and to see if a VTK file was generated correctly. Expectations indicated that it should have been used more often. That being said, Paraview is probably a tool that works best for further development of new features and checking the results for both Focus Konstruksjon and OOCfem, rather than during the actual implementation of the file based system.

Activiz provided a .NET wrapper for VTK. It came a little short on parsing values, when adding precision decimals to the results. Nevertheless, it was not found a better solution for the post-processing phase. Together with Paraview, VTK examples and a live community using VTK and Activiz, it is believed that it was the right choice.

This thesis has been focusing on developing a file based system which could fulfill the requirements in chapter 2.1. Reviewing all functional requirements (FR), it is believed that this thesis has met all these requests to the file based system. Even the requirement of the result database being read by another third-party visualization program (FR11) has been met, which seemed difficult to fulfill in the begining. This thesis has not been able to evaluate the performance requirement in the non-functional requirements. However, it is believed to be fulfillled together with the other non-functional requirements, which has already been achieved, namely; Documentation, Open Sourced and maintainability.

For a more thorough discussion and evaluation of the file formats, it is recommended to see chapter 4.4.

Chapter 8

Conclusion and Further Work

8.1 Conclusion

The file based system with an input file and a result database output file is the main deliverable of this thesis. The file based system is not 100% completed and would not be possible to roll out in the next release for Focus Konstruksjon. However, the basis for the file based system is implemented. With an input file representing geometric and analysis data, which is the data of a pre-process phase, together with an output data file representing stresses and displacements, which is the post-processing phase data. Both formats were successfully implemented for a linear analysis simulation.

Writing tests, documentation, and a list of technical descriptions of future work are also deliverables of this thesis. These deliverables are found in the code base of the new system and in appendix chapters.

Feedback from Focus Software's and OOCfem's stakeholders demonstrates that the results are satisfactory. There are signs of relief from both parts regarding the fact that the file based system is in place, especially considering debugging, maintaining and future developing of Focus Konstruksjon. The software will be even more competitive in the construction market, and provide a better solution for the users. This implies that this thesis has delivered a solution to the problem described in chapter 1.

It would have been difficult to achive this outcome without the help from developers from both Focus Konstruksjon and OOCfem. Such help included eg. a provision of code for writing OOCfem elements to a VTK file, and already written tests for different cases in Focus Konstruksjon. Their ambitions of having a file based system, made it easier to drive the process to meet the requirements and the expectations of this thesis.

Stakeholders of both Focus Konstruksjon and OOCfem are already exploring the opportunities for what could be implemented next, after the file based system is fully implemented.

8.2 Future work

For the linear analysis, the file based system is fully implemented. The next step in integrating the system, is to implement it into other simulation analyeis such as a non-linear analysis and a buckling analysis. Focus Software has six additional analysis algorithms, where the file based system is going to be implemented. The file based system is correctly working on feature branches for both OOCfem and Focus Konstruksjon. These branches have to be carefully merged with the actual solution, to integrate all necessary functionalities, before being able to further develop the system.

The file based system now saves input and output files to the local application folder. To benefit from the ability of not having to run a simulation of a construction twice, Focus Konstruksjon has to implement a check of the results in the output file. If the model in Focus Konstruksjon has undertaken the exact same simulation before, the expensive process of running OOCfem again can be prevented, by showing the results from the result output file. To check this, the system has to be developed to control which analyeis that have been run or not. Today, directories are being stored with wrapper-ID names, which gives an ID for every simulation conducted. These IDs have to be stored and controlled in Focus Konstruksjon to fully benefit the new file based system.

For a more technical description of future work, see appendix chapter C.

Bibliography

[1] Solution provider profiles - cimdata. http://www.cimdata.com/en/newsletter/2013/28/02/28.02.01.html. (Accessed on 28-02-2017).

[2] Parsing - wikipedia. https://en.wikipedia.org/wiki/Parsing. (Accessed on 28-02-2017).

[3] Ascii - wikipedia. https://en.wikipedia.org/wiki/ASCII. (Accessed on 28-02-2017).

[4] Binary gives significant performance advantage (mesh & solve) – cfd online discussion forums. https://www.cfd-online.com/Forums/openfoam/136983-binary-gives-significant-performance-advantage-mesh-solve.html. (Accessed on 28-02-2017).

[5] Bruce Perens et al. The open source definition. *Open sources: voices from the open source revolution*, 1:171–188, 1999.

[6] J Michopoulos, P Mast, T Chwastyk, L Gause, and R Badaliance. Femml for data exchange between fea codes. In *ANSYS Users' Group Conference*, 2001.

[7] Simulation analysis. http://www.jotneit.no/solutions/simulation-analysis. (Accessed on 28-02-2017).

[8] An online fem model converter - nastran - eng-tips. http://www.eng-tips.com/viewthread.cfm?qid=401488. (Accessed on 28-02-2017).

[9] Dette er verdens 10 mest populære programmeringsspråk - tu.no. https://www.tu.no/artikler/dette-er-verdens-10-mest-populaere-programmeringssprak/231139. (Accessed on 28-02-2017).

[10] C++ - wikipedia. https://en.wikipedia.org/wiki/C%2B%2B. (Accessed on 28-02-2017).

[11] .net framework - wikipedia. https://en.wikipedia.org/wiki/.NET_Framework. (Accessed on 28-02-2017).

[12] What is managed code? https://msdn.microsoft.com/en-us/library/windows/desktop/bb318664(v=vs.85).aspx. (Accessed on 28-02-2017).

[13] F sharp (programming language) - wikipedia. https://en.wikipedia.org/wiki/F_Sharp_(programming_language). (Accessed on 28-02-2017).

[14] Nx nastran user's guide. https://docs.plm.automation.siemens.com/data_services/resources/nxnastran/10/help/en_US/tdocExt/pdf/%20User.pdf. (Accessed on 28-02-2017).

[15] Usfos_um_06.pdf. http://usfos.no/manuals/usfos/users/documents/Usfos_UM_06.pdf. (Accessed on 28-02-2017).

[16] Sesam-interface-file.pdf. https://projects.dnvgl.com/sesam/manuals/SESAM-Interface-File-Description/SESAM-Interface-File.pdf. (Accessed on 28-02-2017).

[17] Bjørn Haugen. Cfem input file format. Cfem input file format, 1st ed 2017.

[18] Usfos - reality engineering. http://www.usfos.no/. (Accessed on 28-02-2017).

[19] C Geuzaine and JF Remacle. Gmsh: A three-dimensional finite element mesh generator with built-in pre-and post-processing facilities, version 2.2. 4, 2008.

[20] Why ap209? - eplm interoperability. http://www.ap209.org/why-ap209. (Accessed on 28-02-2017).

[21] Introduction/scope - eplm interoperability. http://www.ap209.org/introduction. (Accessed on 28-02-2017).

[22] Vtk - dev - how to handle fem data in vtk? http://vtk.1045678.n5.nabble.com/how-to-handle-FEM-data-in-VTK-td1253575.html. (Accessed on 28-02-2017).

[23] Medcoupling users' guide: Notes on the med file format. http://docs.salome-platform.org/latest/dev/MEDCoupling/med-file.html. (Accessed on 28-02-2017).

[24] Unidata | netcdf. http://www.unidata.ucar.edu/software/netcdf/. (Accessed on 28-02-2017).

[25] Json - wikipedia. https://en.wikipedia.org/wiki/JSON. (Accessed on 28-02-2017).

[26] Jef claes: Using json serialization outside a web context. http://www.jefclaes.be/2011/02/using-json-serialization-outside-web.html. (Accessed on 28-02-2017).

[27] Hdf group - hdf5. https://support.hdfgroup.org/HDF5/. (Accessed on 28-02-2017).

[28] Hdf and .net project. https://support.hdfgroup.org/projects/hdf.net/. (Accessed on 28-02-2017).

[29] Hdfgroup/hdf.pinvoke: Raw hdf5 power for .net. https://github.com/HDFGroup/HDF.PInvoke. (Accessed on 28-02-2017).

[30] Vtk - the visualization toolkit. http://www.vtk.org/. (Accessed on 28-02-2017).

[31] Vtk/csharp/activiz.net - kitwarepublic. http://www.vtk.org/Wiki/VTK/CSharp/ActiViz.NET. (Accessed on 28-02-2017).

[32] Vtk file formats — earth models. http://www.earthmodels.org/software/vtk-and-paraview/vtk-file-formats. (Accessed on 28-02-2017).

[33] Paraview. http://www.paraview.org/. (Accessed on 28-02-2017).

[34] Test-driven development - wikipedia. https://en.wikipedia.org/wiki/Test-driven_development. (Accessed on 07-03-2017).

[35] Unit test basics. https://msdn.microsoft.com/en-us/library/hh694602.aspx. (Accessed on 07-03-2017).

[36] Arrange Act Assert. [online]. *http://wiki.c2.com/?ArrangeActAssert*, 2016. (Accessed on 07-03-2017).

[37] E Michael Maximilien and Laurie Williams. Assessing test-driven development at ibm. In *Software Engineering, 2003. Proceedings. 25th International Conference on*, pages 564–569. IEEE, 2003.

[38] Hakan Erdogmus, Maurizio Morisio, and Marco Torchiano. On the effectiveness of the test-first approach to programming. *IEEE Transactions on software Engineering*, 31(3):226–237, 2005.

[39] If and when you should use test-driven development | zeroturnaround.com. https://zeroturnaround.com/rebellabs/if-and-when-you-should-use-test-driven-development/. (Accessed on 08-03-2017).

[40] John Deacon. Model-view-controller (mvc) architecture. *Online][Citado em: 10 de março de 2006.] http://www. jdl. co. uk/briefings/MVC. pdf*, 2009.

[41] Robert T Monroe, Andrew Kompanek, Ralph Melton, and David Garlan. Architectural styles, design patterns, and objects. *IEEE software*, 14(1):43–52, 1997.

[42] Javabeans - wikipedia. https://en.wikipedia.org/wiki/JavaBeans. (Accessed on 02-06-2017).

[43] App data storage - windows app development. https://msdn.microsoft.com/en-us/library/windows/apps/hh464917.aspx. (Accessed on 24-05-2017).

[44] Nuget - wikipedia. https://en.wikipedia.org/wiki/NuGet. (Accessed on 24-05-2017).

[45] Jesus-driven development. http://no-kill-switch.ghost.io/jesus-driven-development/. (Accessed on 02-06-2017).

[46] Dumpster engagement, faith-driven development, go native, hired scapegoat, 'jesus', non-solicitation agreement | jargon from the world of technology consulting | informit. http://www.informit.com/articles/article.aspx?p=1381169&seqNum=2. (Accessed on 02-06-2017).

[47] Object-oriented programming - wikipedia. https://en.wikipedia.org/wiki/Object-oriented_programming. (Accessed on 21-04-2017).

[48] Robert C Martin. *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009.

[49] Solid (object-oriented design) - wikipedia. https://en.wikipedia.org/wiki/SOLID_(object-oriented_design). (Accessed on 24-05-2017).

[50] Xml - wikipedia. https://en.wikipedia.org/wiki/XML. (Accessed on 28-02-2017).

[51] Femtools interface and driver programs. http://www.femtools.com/products/ftinterfaces.htm. (Accessed on 28-02-2017).

[52] Universal file format - wikipedia. https://en.wikipedia.org/wiki/Universal_File_Format. (Accessed on 28-02-2017).

[53] Free mechanical engineering: Finite element analysis. http://www.freebyte.com/cad/fea.htm. (Accessed on 28-02-2017).

[54] Aladdin matrix and finite element environment. http://www.isr.umd.edu/~austin/aladdin.html#sec6. (Accessed on 28-02-2017).

[55] Abaqus 2016 documentation. http://50.16.225.63/v2016/. (Accessed on 28-02-2017).

[56] Abaqus scripting reference guide (2016). http://50.16.225.63/v2016/books/ker/default.htm. (Accessed on 28-02-2017).

[57] pynastran_trunk.pdf. https://media.readthedocs.org/pdf/pynastran_trunk/latest/pynastran_trunk.pdf. (Accessed on 28-02-2017).

Appendix A

Acronyms

FE	Finite Element
FEA	Finite element analysis
FEM	Finite Element Method
API	Application Programming Interface
DLL	Dynamic-Link Library
CAD	Computer Aided Design
CAE	Computer Aided Engineering
MVC	Model View Controller
XML	Extensible Markup Language

Appendix B

File Formats - USFOS and VTK

B.1 USFOS

Usfos format devided in three to get the best representation of the file in the paper format of this thesis. The file represents a eight meter long beam, fixed in both ends with only its structural weight as forces.

Excerpt from USFOS documentation

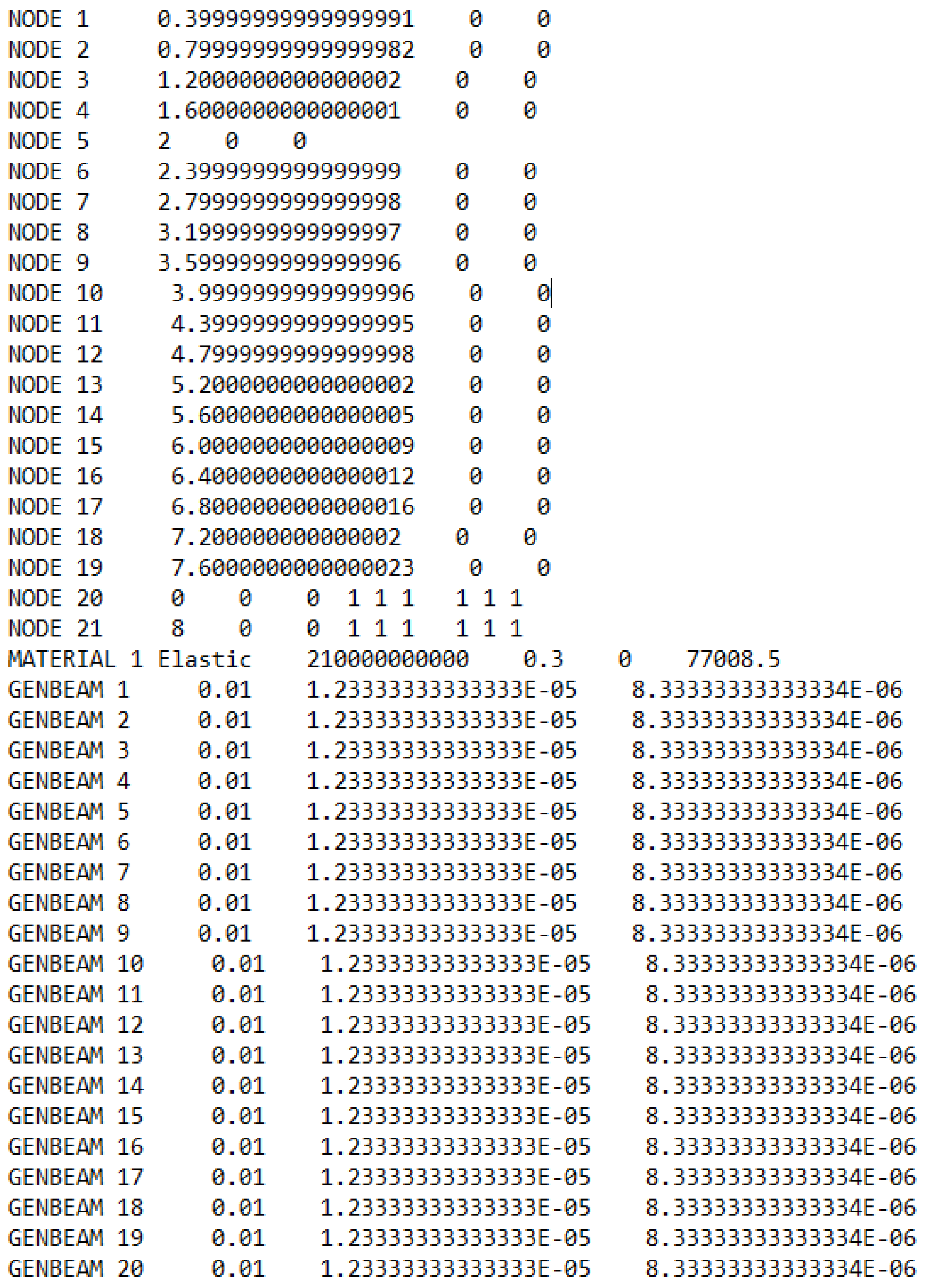
The extensions of the Usfos format, in terms of new components or additional fields in already existing components.

B.2 VTK

VTK format devided in three to get the best representation of the file in the paper format of this thesis.

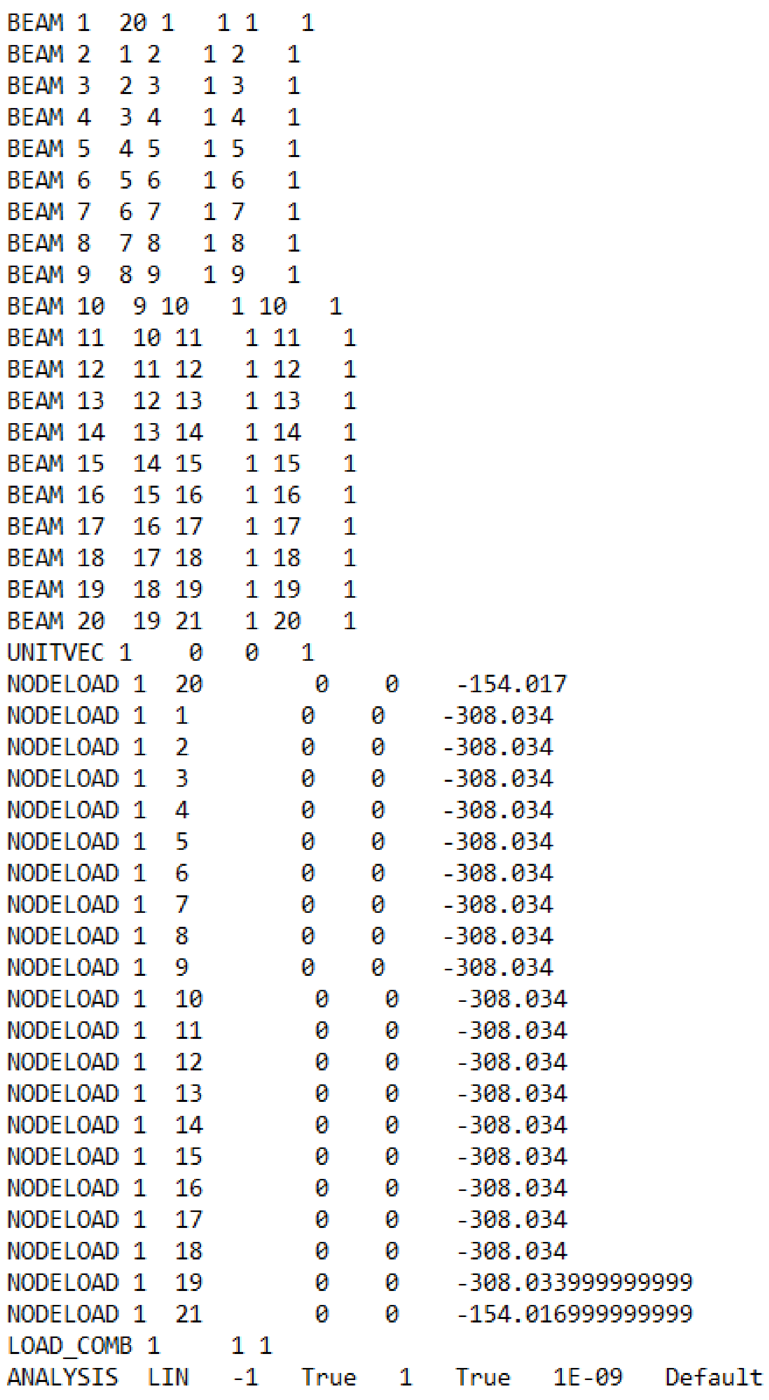
VTK binary

Illustrates the VTK generated as a binary file.



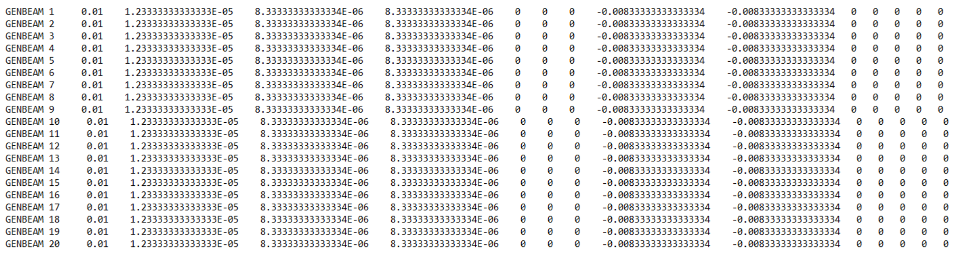
```
NODE 1     0.39999999999999991    0    0
NODE 2     0.79999999999999982    0    0
NODE 3     1.2000000000000002    0    0
NODE 4     1.6000000000000001    0    0
NODE 5     2    0    0
NODE 6     2.3999999999999999    0    0
NODE 7     2.7999999999999998    0    0
NODE 8     3.1999999999999997    0    0
NODE 9     3.5999999999999996    0    0
NODE 10     3.9999999999999996    0    0
NODE 11     4.3999999999999995    0    0
NODE 12     4.7999999999999998    0    0
NODE 13     5.2000000000000002    0    0
NODE 14     5.6000000000000005    0    0
NODE 15     6.0000000000000009    0    0
NODE 16     6.4000000000000012    0    0
NODE 17     6.8000000000000016    0    0
NODE 18     7.200000000000002    0    0
NODE 19     7.6000000000000023    0    0
NODE 20     0    0    0  1 1 1   1 1 1
NODE 21     8    0    0  1 1 1   1 1 1
MATERIAL 1 Elastic    210000000000    0.3    0    77008.5
GENBEAM 1     0.01    1.23333333333333E-05    8.33333333333334E-06
GENBEAM 2     0.01    1.23333333333333E-05    8.33333333333334E-06
GENBEAM 3     0.01    1.23333333333333E-05    8.33333333333334E-06
GENBEAM 4     0.01    1.23333333333333E-05    8.33333333333334E-06
GENBEAM 5     0.01    1.23333333333333E-05    8.33333333333334E-06
GENBEAM 6     0.01    1.23333333333333E-05    8.33333333333334E-06
GENBEAM 7     0.01    1.23333333333333E-05    8.33333333333334E-06
GENBEAM 8     0.01    1.23333333333333E-05    8.33333333333334E-06
GENBEAM 9     0.01    1.23333333333333E-05    8.33333333333334E-06
GENBEAM 10     0.01    1.23333333333333E-05    8.33333333333334E-06
GENBEAM 11     0.01    1.23333333333333E-05    8.33333333333334E-06
GENBEAM 12     0.01    1.23333333333333E-05    8.33333333333334E-06
GENBEAM 13     0.01    1.23333333333333E-05    8.33333333333334E-06
GENBEAM 14     0.01    1.23333333333333E-05    8.33333333333334E-06
GENBEAM 15     0.01    1.23333333333333E-05    8.33333333333334E-06
GENBEAM 16     0.01    1.23333333333333E-05    8.33333333333334E-06
GENBEAM 17     0.01    1.23333333333333E-05    8.33333333333334E-06
GENBEAM 18     0.01    1.23333333333333E-05    8.33333333333334E-06
GENBEAM 19     0.01    1.23333333333333E-05    8.33333333333334E-06
GENBEAM 20     0.01    1.23333333333333E-05    8.33333333333334E-06
```

Figure B.1: Usfos format top



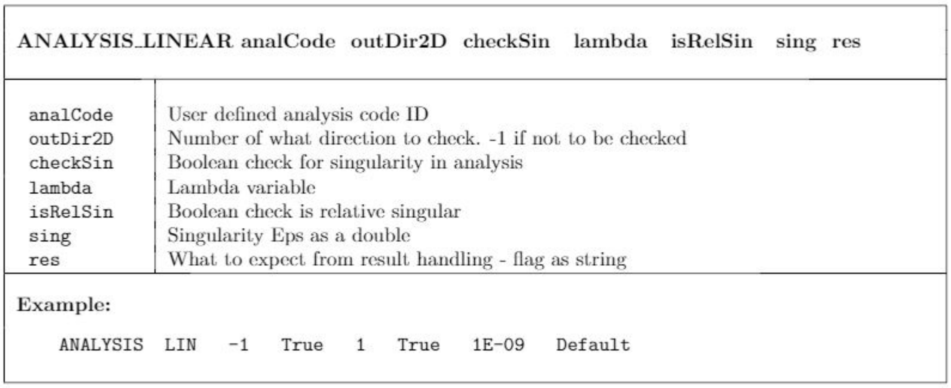
```
BEAM 1  20 1   1 1   1
BEAM 2  1 2   1 2   1
BEAM 3  2 3   1 3   1
BEAM 4  3 4   1 4   1
BEAM 5  4 5   1 5   1
BEAM 6  5 6   1 6   1
BEAM 7  6 7   1 7   1
BEAM 8  7 8   1 8   1
BEAM 9  8 9   1 9   1
BEAM 10  9 10   1 10   1
BEAM 11  10 11   1 11   1
BEAM 12  11 12   1 12   1
BEAM 13  12 13   1 13   1
BEAM 14  13 14   1 14   1
BEAM 15  14 15   1 15   1
BEAM 16  15 16   1 16   1
BEAM 17  16 17   1 17   1
BEAM 18  17 18   1 18   1
BEAM 19  18 19   1 19   1
BEAM 20  19 21   1 20   1
UNITVEC 1    0   0   1
NODELOAD 1  20       0    0    -154.017
NODELOAD 1  1       0    0    -308.034
NODELOAD 1  2       0    0    -308.034
NODELOAD 1  3       0    0    -308.034
NODELOAD 1  4       0    0    -308.034
NODELOAD 1  5       0    0    -308.034
NODELOAD 1  6       0    0    -308.034
NODELOAD 1  7       0    0    -308.034
NODELOAD 1  8       0    0    -308.034
NODELOAD 1  9       0    0    -308.034
NODELOAD 1  10       0    0    -308.034
NODELOAD 1  11       0    0    -308.034
NODELOAD 1  12       0    0    -308.034
NODELOAD 1  13       0    0    -308.034
NODELOAD 1  14       0    0    -308.034
NODELOAD 1  15       0    0    -308.034
NODELOAD 1  16       0    0    -308.034
NODELOAD 1  17       0    0    -308.034
NODELOAD 1  18       0    0    -308.034
NODELOAD 1  19       0    0    -308.033999999999
NODELOAD 1  21       0    0    -154.016999999999
LOAD_COMB 1     1 1
ANALYSIS  LIN   -1   True   1   True   1E-09   Default
```

Figure B.2: Usfos format bottom



```
GENBEAM 1     0.01    1.23333333333333E-05    8.33333333333334E-06    8.33333333333334E-06    0    0    0    -0.00833333333333334    -0.00833333333333334   0   0   0   0   0
GENBEAM 2     0.01    1.23333333333333E-05    8.33333333333334E-06    8.33333333333334E-06    0    0    0    -0.00833333333333334    -0.00833333333333334   0   0   0   0   0
GENBEAM 3     0.01    1.23333333333333E-05    8.33333333333334E-06    8.33333333333334E-06    0    0    0    -0.00833333333333334    -0.00833333333333334   0   0   0   0   0
GENBEAM 4     0.01    1.23333333333333E-05    8.33333333333334E-06    8.33333333333334E-06    0    0    0    -0.00833333333333334    -0.00833333333333334   0   0   0   0   0
GENBEAM 5     0.01    1.23333333333333E-05    8.33333333333334E-06    8.33333333333334E-06    0    0    0    -0.00833333333333334    -0.00833333333333334   0   0   0   0   0
GENBEAM 6     0.01    1.23333333333333E-05    8.33333333333334E-06    8.33333333333334E-06    0    0    0    -0.00833333333333334    -0.00833333333333334   0   0   0   0   0
GENBEAM 7     0.01    1.23333333333333E-05    8.33333333333334E-06    8.33333333333334E-06    0    0    0    -0.00833333333333334    -0.00833333333333334   0   0   0   0   0
GENBEAM 8     0.01    1.23333333333333E-05    8.33333333333334E-06    8.33333333333334E-06    0    0    0    -0.00833333333333334    -0.00833333333333334   0   0   0   0   0
GENBEAM 9     0.01    1.23333333333333E-05    8.33333333333334E-06    8.33333333333334E-06    0    0    0    -0.00833333333333334    -0.00833333333333334   0   0   0   0   0
GENBEAM 10     0.01    1.23333333333333E-05    8.33333333333334E-06    8.33333333333334E-06    0    0    0    -0.00833333333333334    -0.00833333333333334   0   0   0   0   0
GENBEAM 11     0.01    1.23333333333333E-05    8.33333333333334E-06    8.33333333333334E-06    0    0    0    -0.00833333333333334    -0.00833333333333334   0   0   0   0   0
GENBEAM 12     0.01    1.23333333333333E-05    8.33333333333334E-06    8.33333333333334E-06    0    0    0    -0.00833333333333334    -0.00833333333333334   0   0   0   0   0
GENBEAM 13     0.01    1.23333333333333E-05    8.33333333333334E-06    8.33333333333334E-06    0    0    0    -0.00833333333333334    -0.00833333333333334   0   0   0   0   0
GENBEAM 14     0.01    1.23333333333333E-05    8.33333333333334E-06    8.33333333333334E-06    0    0    0    -0.00833333333333334    -0.00833333333333334   0   0   0   0   0
GENBEAM 15     0.01    1.23333333333333E-05    8.33333333333334E-06    8.33333333333334E-06    0    0    0    -0.00833333333333334    -0.00833333333333334   0   0   0   0   0
GENBEAM 16     0.01    1.23333333333333E-05    8.33333333333334E-06    8.33333333333334E-06    0    0    0    -0.00833333333333334    -0.00833333333333334   0   0   0   0   0
GENBEAM 17     0.01    1.23333333333333E-05    8.33333333333334E-06    8.33333333333334E-06    0    0    0    -0.00833333333333334    -0.00833333333333334   0   0   0   0   0
GENBEAM 18     0.01    1.23333333333333E-05    8.33333333333334E-06    8.33333333333334E-06    0    0    0    -0.00833333333333334    -0.00833333333333334   0   0   0   0   0
GENBEAM 19     0.01    1.23333333333333E-05    8.33333333333334E-06    8.33333333333334E-06    0    0    0    -0.00833333333333334    -0.00833333333333334   0   0   0   0   0
GENBEAM 20     0.01    1.23333333333333E-05    8.33333333333334E-06    8.33333333333334E-06    0    0    0    -0.00833333333333334    -0.00833333333333334   0   0   0   0   0
```

Figure B.3: Usfos format genbeam - whole



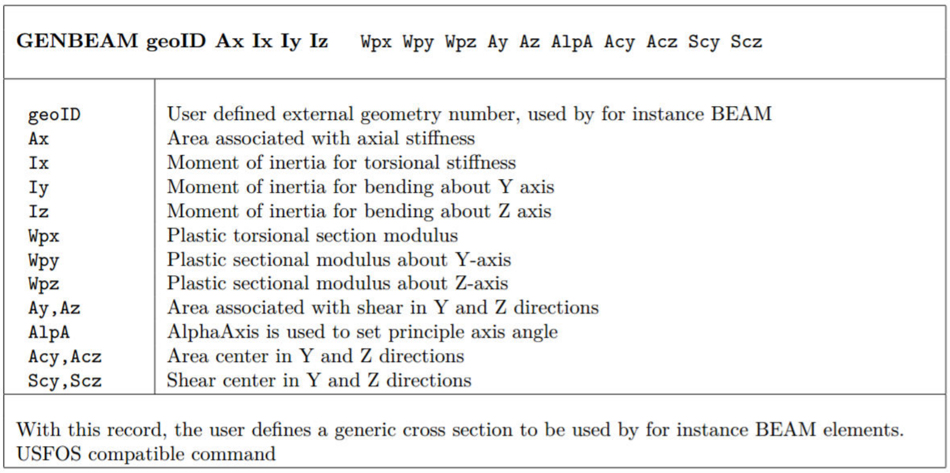
ANALYSIS_LINEAR analCode outDir2D checkSin lambda isRelSin sing res

`analCode`	User defined analysis code ID
`outDir2D`	Number of what direction to check. -1 if not to be checked
`checkSin`	Boolean check for singularity in analysis
`lambda`	Lambda variable
`isRelSin`	Boolean check is relative singular
`sing`	Singularity Eps as a double
`res`	What to expect from result handling - flag as string

Example:

```
ANALYSIS  LIN   -1   True   1   True   1E-09   Default
```

Figure B.4: Usfos documentation ANALYSIS_LINEAR

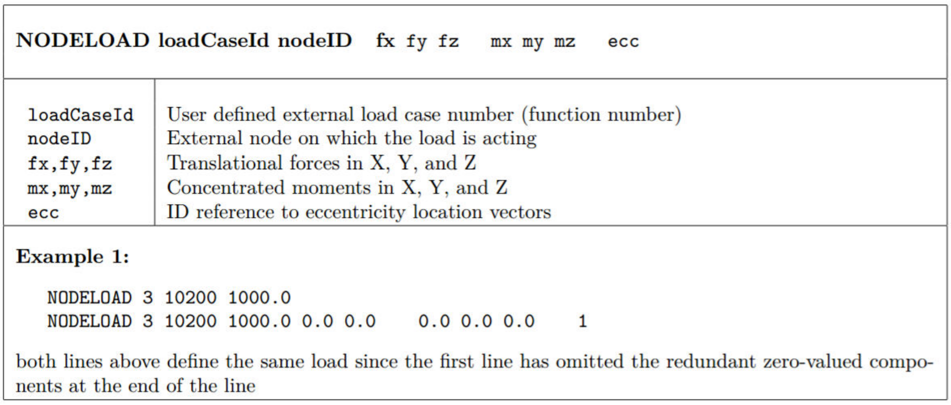


GENBEAM geoID Ax Ix Iy Iz `Wpx Wpy Wpz Ay Az AlpA Acy Acz Scy Scz`

`geoID`	User defined external geometry number, used by for instance BEAM
`Ax`	Area associated with axial stiffness
`Ix`	Moment of inertia for torsional stiffness
`Iy`	Moment of inertia for bending about Y axis
`Iz`	Moment of inertia for bending about Z axis
`Wpx`	Plastic torsional section modulus
`Wpy`	Plastic sectional modulus about Y-axis
`Wpz`	Plastic sectional modulus about Z-axis
`Ay,Az`	Area associated with shear in Y and Z directions
`AlpA`	AlphaAxis is used to set principle axis angle
`Acy,Acz`	Area center in Y and Z directions
`Scy,Scz`	Shear center in Y and Z directions

With this record, the user defines a generic cross section to be used by for instance BEAM elements.
USFOS compatible command

Figure B.5: Usfos documentation - GENBEAM



NODELOAD loadCaseId nodeID `fx fy fz` `mx my mz` `ecc`

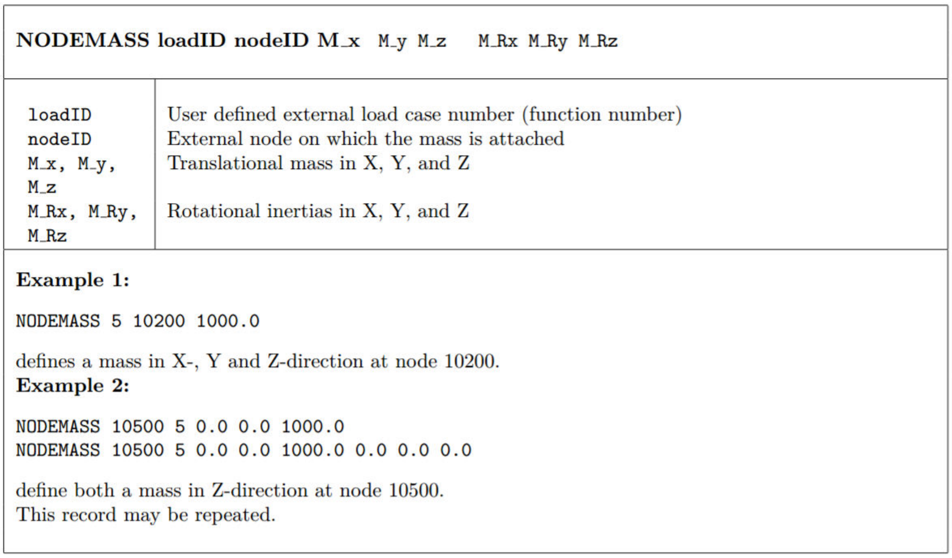
`loadCaseId`	User defined external load case number (function number)
`nodeID`	External node on which the load is acting
`fx,fy,fz`	Translational forces in X, Y, and Z
`mx,my,mz`	Concentrated moments in X, Y, and Z
`ecc`	ID reference to eccentricity location vectors

Example 1:

```
NODELOAD 3 10200 1000.0
NODELOAD 3 10200 1000.0 0.0 0.0    0.0 0.0 0.0    1
```

both lines above define the same load since the first line has omitted the redundant zero-valued components at the end of the line

Figure B.6: Usfos documentation - NODELOAD



NODEMASS loadID nodeID M_x `M_y M_z` `M_Rx M_Ry M_Rz`

`loadID`	User defined external load case number (function number)
`nodeID`	External node on which the mass is attached
`M_x, M_y, M_z`	Translational mass in X, Y, and Z
`M_Rx, M_Ry, M_Rz`	Rotational inertias in X, Y, and Z

Example 1:

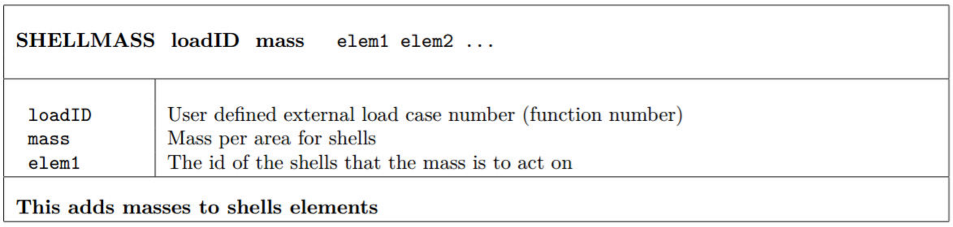
```
NODEMASS 5 10200 1000.0
```

defines a mass in X-, Y and Z-direction at node 10200.
Example 2:

```
NODEMASS 10500 5 0.0 0.0 1000.0
NODEMASS 10500 5 0.0 0.0 1000.0 0.0 0.0 0.0
```

define both a mass in Z-direction at node 10500.
This record may be repeated.

Figure B.7: Usfos documentation - NODEMASS

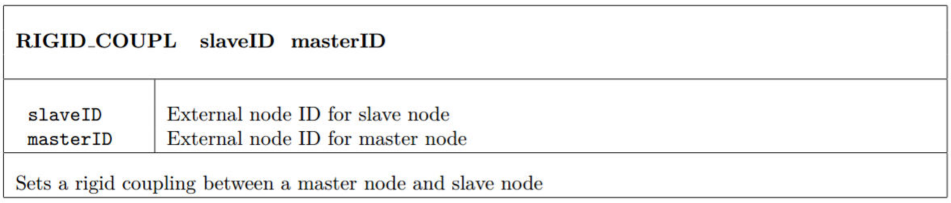


SHELLMASS loadID mass `elem1 elem2 ...`

`loadID`	User defined external load case number (function number)
`mass`	Mass per area for shells
`elem1`	The id of the shells that the mass is to act on

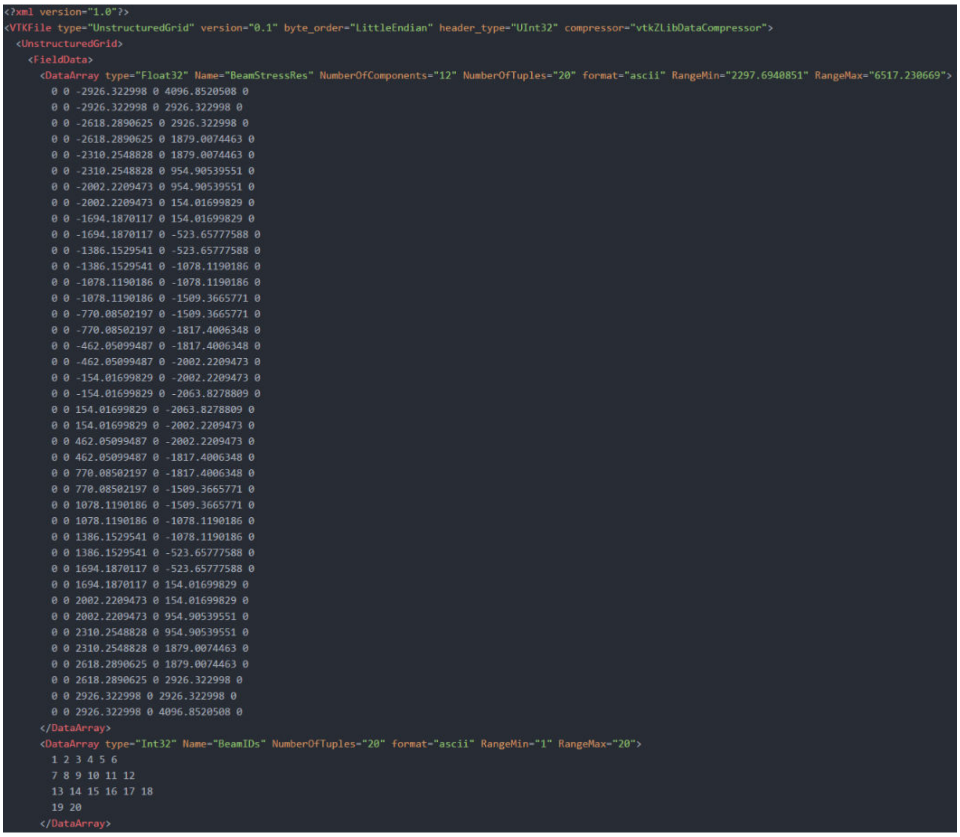
This adds masses to shells elements

Figure B.8: Usfos documentation - SHELLMASS



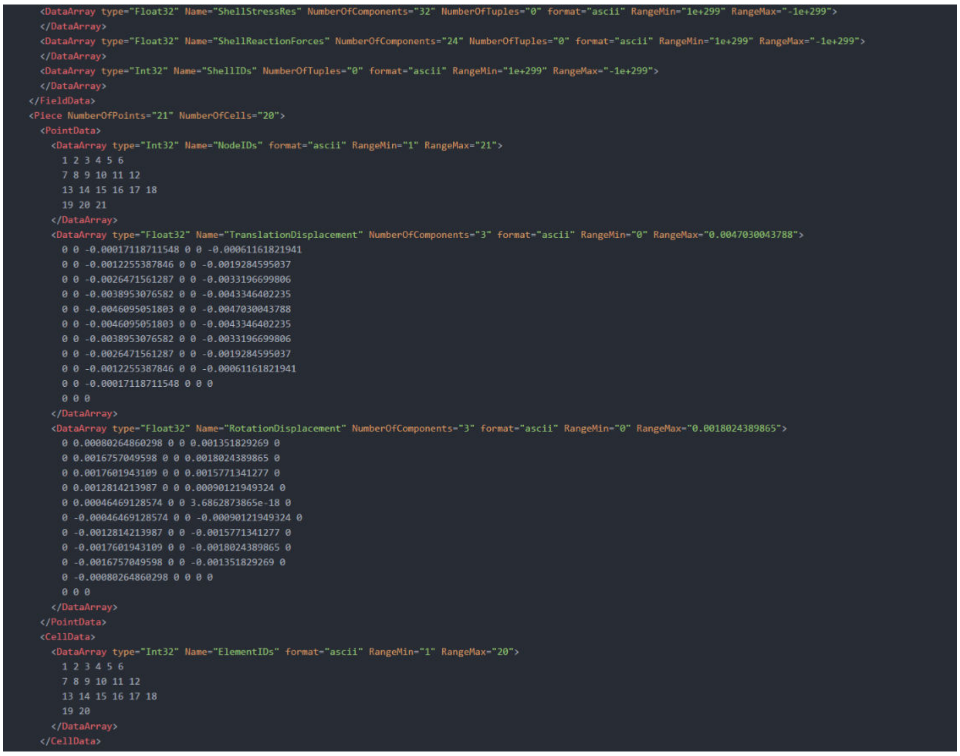
RIGID_COUPL slaveID masterID	
`slaveID`	External node ID for slave node
`masterID`	External node ID for master node
Sets a rigid coupling between a master node and slave node	

Figure B.9: Usfos documentation - RIGID_COUPL



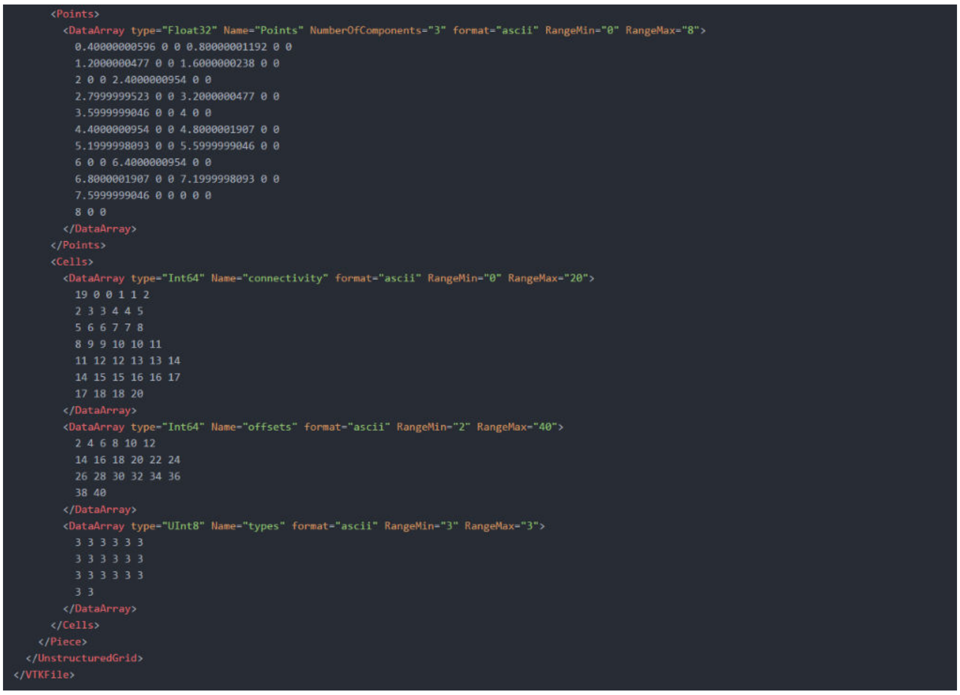
```
<?xml version="1.0"?>
<VTKFile type="UnstructuredGrid" version="0.1" byte_order="LittleEndian" header_type="UInt32" compressor="vtkZLibDataCompressor">
  <UnstructuredGrid>
    <FieldData>
      <DataArray type="Float32" Name="BeamStressRes" NumberOfComponents="12" NumberOfTuples="20" format="ascii" RangeMin="2297.6940851" RangeMax="6517.230669">
        0 0 -2926.322998 0 4096.8520508 0
        0 0 -2926.322998 0 2926.322998 0
        0 0 -2618.2890625 0 2926.322998 0
        0 0 -2618.2890625 0 1879.0074463 0
        0 0 -2310.2548828 0 1879.0074463 0
        0 0 -2310.2548828 0 954.90539551 0
        0 0 -2002.2209473 0 954.90539551 0
        0 0 -2002.2209473 0 154.01699829 0
        0 0 -1694.1870117 0 154.01699829 0
        0 0 -1694.1870117 0 -523.65777588 0
        0 0 -1386.1529541 0 -523.65777588 0
        0 0 -1386.1529541 0 -1078.1190186 0
        0 0 -1078.1190186 0 -1078.1190186 0
        0 0 -1078.1190186 0 -1509.3665771 0
        0 0 -770.08502197 0 -1509.3665771 0
        0 0 -770.08502197 0 -1817.4006348 0
        0 0 -462.05099487 0 -1817.4006348 0
        0 0 -462.05099487 0 -2002.2209473 0
        0 0 -154.01699829 0 -2002.2209473 0
        0 0 -154.01699829 0 -2063.8278809 0
        0 0 154.01699829 0 -2063.8278809 0
        0 0 154.01699829 0 -2002.2209473 0
        0 0 462.05099487 0 -2002.2209473 0
        0 0 462.05099487 0 -1817.4006348 0
        0 0 770.08502197 0 -1817.4006348 0
        0 0 770.08502197 0 -1509.3665771 0
        0 0 1078.1190186 0 -1509.3665771 0
        0 0 1078.1190186 0 -1078.1190186 0
        0 0 1386.1529541 0 -1078.1190186 0
        0 0 1386.1529541 0 -523.65777588 0
        0 0 1694.1870117 0 -523.65777588 0
        0 0 1694.1870117 0 154.01699829 0
        0 0 2002.2209473 0 154.01699829 0
        0 0 2002.2209473 0 954.90539551 0
        0 0 2310.2548828 0 954.90539551 0
        0 0 2310.2548828 0 1879.0074463 0
        0 0 2618.2890625 0 1879.0074463 0
        0 0 2618.2890625 0 2926.322998 0
        0 0 2926.322998 0 2926.322998 0
        0 0 2926.322998 0 4096.8520508 0
      </DataArray>
      <DataArray type="Int32" Name="BeamIDs" NumberOfTuples="20" format="ascii" RangeMin="1" RangeMax="20">
        1 2 3 4 5 6
        7 8 9 10 11 12
        13 14 15 16 17 18
        19 20
      </DataArray>
```

Figure B.10: VTK format top



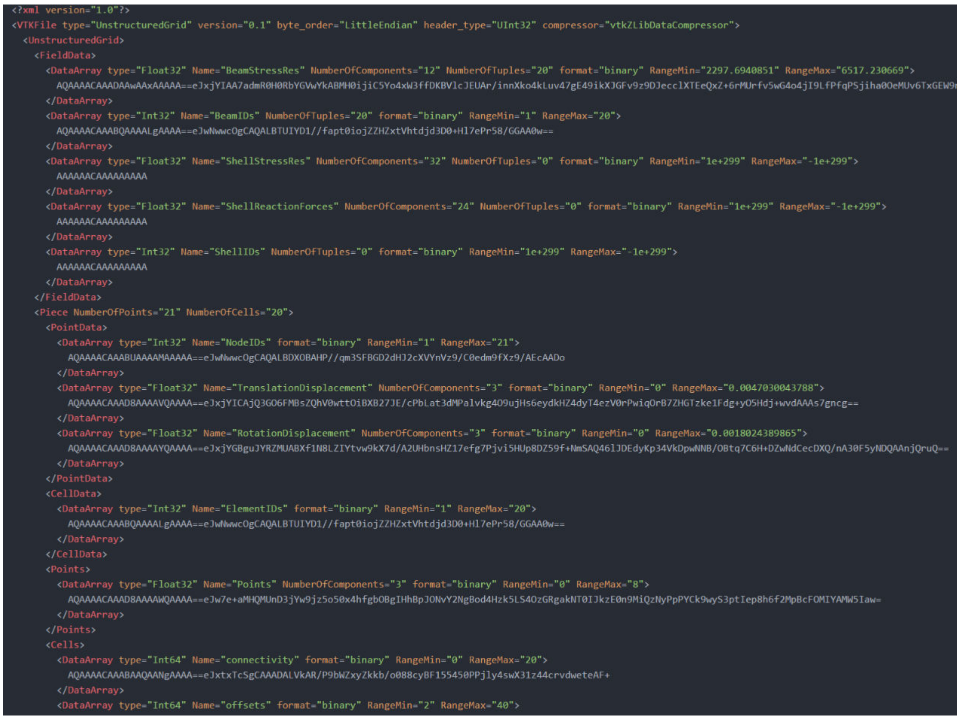
```
    <DataArray type="Float32" Name="ShellStressRes" NumberOfComponents="32" NumberOfTuples="0" format="ascii" RangeMin="1e+299" RangeMax="-1e+299">
    </DataArray>
    <DataArray type="Float32" Name="ShellReactionForces" NumberOfComponents="24" NumberOfTuples="0" format="ascii" RangeMin="1e+299" RangeMax="-1e+299">
    </DataArray>
    <DataArray type="Int32" Name="ShellIDs" NumberOfTuples="0" format="ascii" RangeMin="1e+299" RangeMax="-1e+299">
    </DataArray>
  </FieldData>
  <Piece NumberOfPoints="21" NumberOfCells="20">
    <PointData>
      <DataArray type="Int32" Name="NodeIDs" format="ascii" RangeMin="1" RangeMax="21">
        1 2 3 4 5 6
        7 8 9 10 11 12
        13 14 15 16 17 18
        19 20 21
      </DataArray>
      <DataArray type="Float32" Name="TranslationDisplacement" NumberOfComponents="3" format="ascii" RangeMin="0" RangeMax="0.0047030043788">
        0 0 -0.00017118711548 0 0 -0.00061161821941
        0 0 -0.0012255387846 0 0 -0.0019284595037
        0 0 -0.0026471561287 0 0 -0.0033196699806
        0 0 -0.0038953076582 0 0 -0.0043346402235
        0 0 -0.0046095051803 0 0 -0.0047030043788
        0 0 -0.0046095051803 0 0 -0.0043346402235
        0 0 -0.0038953076582 0 0 -0.0033196699806
        0 0 -0.0026471561287 0 0 -0.0019284595037
        0 0 -0.0012255387846 0 0 -0.00061161821941
        0 0 -0.00017118711548 0 0 0
        0 0 0
      </DataArray>
      <DataArray type="Float32" Name="RotationDisplacement" NumberOfComponents="3" format="ascii" RangeMin="0" RangeMax="0.0018024389865">
        0 0.00080264860298 0 0 0.001351829269 0
        0 0.0016757049598 0 0 0.0018024389865 0
        0 0.0017601943109 0 0 0.0015771341277 0
        0 0.0012814213987 0 0 0.00090121949324 0
        0 0.00046469128574 0 0 3.6862873865e-18 0
        0 -0.00046469128574 0 0 -0.00090121949324 0
        0 -0.0012814213987 0 0 -0.0015771341277 0
        0 -0.0017601943109 0 0 -0.0018024389865 0
        0 -0.0016757049598 0 0 -0.001351829269 0
        0 -0.00080264860298 0 0 0 0
        0 0 0
      </DataArray>
    </PointData>
    <CellData>
      <DataArray type="Int32" Name="ElementIDs" format="ascii" RangeMin="1" RangeMax="20">
        1 2 3 4 5 6
        7 8 9 10 11 12
        13 14 15 16 17 18
        19 20
      </DataArray>
    </CellData>
```

Figure B.11: VTK format middle



```
        <Points>
          <DataArray type="Float32" Name="Points" NumberOfComponents="3" format="ascii" RangeMin="0" RangeMax="8">
            0.40000000596 0 0 0.80000001192 0 0
            1.2000000477 0 0 1.6000000238 0 0
            2 0 0 2.4000000954 0 0
            2.7999999523 0 0 3.2000000477 0 0
            3.5999999046 0 0 4 0 0
            4.4000000954 0 0 4.8000001907 0 0
            5.1999998093 0 0 5.5999999046 0 0
            6 0 0 6.4000000954 0 0
            6.8000001907 0 0 7.1999998093 0 0
            7.5999999046 0 0 0 0 0
            8 0 0
          </DataArray>
        </Points>
        <Cells>
          <DataArray type="Int64" Name="connectivity" format="ascii" RangeMin="0" RangeMax="20">
            19 0 0 1 1 2
            2 3 3 4 4 5
            5 6 6 7 7 8
            8 9 9 10 10 11
            11 12 12 13 13 14
            14 15 15 16 16 17
            17 18 18 20
          </DataArray>
          <DataArray type="Int64" Name="offsets" format="ascii" RangeMin="2" RangeMax="40">
            2 4 6 8 10 12
            14 16 18 20 22 24
            26 28 30 32 34 36
            38 40
          </DataArray>
          <DataArray type="UInt8" Name="types" format="ascii" RangeMin="3" RangeMax="3">
            3 3 3 3 3 3
            3 3 3 3 3 3
            3 3 3 3 3 3
            3 3
          </DataArray>
        </Cells>
      </Piece>
    </UnstructuredGrid>
</VTKFile>
```

Figure B.12: VTK format bottom



```
<?xml version="1.0"?>
<VTKFile type="UnstructuredGrid" version="0.1" byte_order="LittleEndian" header_type="UInt32" compressor="vtkZLibDataCompressor">
  <UnstructuredGrid>
    <FieldData>
      <DataArray type="Float32" Name="BeamStressRes" NumberOfComponents="12" NumberOfTuples="20" format="binary" RangeMin="2297.6940851" RangeMax="6517.230669">
        AQAAAACAAADAAwAAxAAAAA==eJxjYIAA7admR0H0RbYGVwYkABMH0ijiC5Yo4xW3ffDKBVlcJEUAr/innXko4kLuv47gE49ikXJGFv9z9DJecclXTEeQxZ+6rMUrfv5wG4o4jI9LfPfqPSjiha0OeMUv6TxGEW9r
      </DataArray>
      <DataArray type="Int32" Name="BeamIDs" NumberOfTuples="20" format="binary" RangeMin="1" RangeMax="20">
        AQAAAACAAABQAAAALgAAAA==eJwNwwcOgCAQALBTUIYD1//fapt0iojZZHZxtVhtdjd3D0+Hl7ePr58/GGAA0w==
      </DataArray>
      <DataArray type="Float32" Name="ShellStressRes" NumberOfComponents="32" NumberOfTuples="0" format="binary" RangeMin="1e+299" RangeMax="-1e+299">
        AAAAAACAAAAAAAAA
      </DataArray>
      <DataArray type="Float32" Name="ShellReactionForces" NumberOfComponents="24" NumberOfTuples="0" format="binary" RangeMin="1e+299" RangeMax="-1e+299">
        AAAAAACAAAAAAAAA
      </DataArray>
      <DataArray type="Int32" Name="ShellIDs" NumberOfTuples="0" format="binary" RangeMin="1e+299" RangeMax="-1e+299">
        AAAAAACAAAAAAAAA
      </DataArray>
    </FieldData>
    <Piece NumberOfPoints="21" NumberOfCells="20">
      <PointData>
        <DataArray type="Int32" Name="NodeIDs" format="binary" RangeMin="1" RangeMax="21">
          AQAAAACAAABUAAAAMAAAAA==eJwNwwcOgCAQALBDXOBAHP//qm3SFBGD2dHJ2cXVYnVz9/C0edm9fXz9/AEcAADo
        </DataArray>
        <DataArray type="Float32" Name="TranslationDisplacement" NumberOfComponents="3" format="binary" RangeMin="0" RangeMax="0.0047030043788">
          AQAAAACAAAD8AAAAVQAAAA==eJxjYICAjQ3GO6FMBsZQhV0wttOiBXB27JE/cPbLat3dMPalvkg4O9ujHs6eydkHZ4dyT4ezV0rPwiqOrB7ZHGTzke1Fdg+yO5Hdj+wvdAAAs7gncg==
        </DataArray>
        <DataArray type="Float32" Name="RotationDisplacement" NumberOfComponents="3" format="binary" RangeMin="0" RangeMax="0.0018024389865">
          AQAAAACAAAD8AAAAYQAAAA==eJxjYGBguJYRZMUABXf1N8LZIYtvw9kX7d/A2UHbnsHZ17efg7Pjvi5HUp8DZ59f+NmSAQ46lJDEdyKp34VkDpwNNB/OBtq7C6H+DZwNdCecDXQ/nA30F5yNDQAAnjQruQ==
        </DataArray>
      </PointData>
      <CellData>
        <DataArray type="Int32" Name="ElementIDs" format="binary" RangeMin="1" RangeMax="20">
          AQAAAACAAABQAAAALgAAAA==eJwNwwcOgCAQALBTUIYD1//fapt0iojZZHZxtVhtdjd3D0+Hl7ePr58/GGAA0w==
        </DataArray>
      </CellData>
      <Points>
        <DataArray type="Float32" Name="Points" NumberOfComponents="3" format="binary" RangeMin="0" RangeMax="8">
          AQAAAACAAAD8AAAAWQAAAA==eJw7e+aMHQMUnD3jYw9jz5o50x4hfgbOBgIHhBpJONvY2NgBod4Hzk5LS4OzGRgakNT0IJkzE0n9MiQzNyPpPYCk9wyS3ptIep8h6f2MpBcFOMIYAMW5Iaw=
        </DataArray>
      </Points>
      <Cells>
        <DataArray type="Int64" Name="connectivity" format="binary" RangeMin="0" RangeMax="20">
          AQAAAACAAABAAQAANgAAAA==eJxtxTcSgCAAADALVkAR/P9bWZxyZkkb/o088cyBF155450PPjly4swX31z44crvdweteAF+
        </DataArray>
        <DataArray type="Int64" Name="offsets" format="binary" RangeMin="2" RangeMax="40">
```

Figure B.13: VTK binary

Appendix C

Further work of Focus Konstruksjon - Technical List

The experience acquired during the development of the file based system has value for Focus Software. The overall general thoughts for further development has been shared in chapter 8. This chapter shows what technical code-specific work that has to be done to complete the implementation of the files based system.

List of further work

- Future work on Focus Konstruksjon side:
  - Structure of directories and files. Now all files are stored in a directory with the ID from the current wrapper as directory name. This ID has to be saved in a dictionary/hash table together with the fkon model file to be able to search for results later on. In CfemPlugin.cs it is suggested to find the name of the fkon file and replace the variable konstruksjonFileName. The best solution for the variable is to write the path "//Focus Software//Focus Konstruksjon" + the fkon file name. This will make sure that the files are written

to the correct appData folder.

- Writing cross sections for shell, the addProperty() method in the cfemWrapper, only writes one unique cross section per wrapper. But for beam cross sections the check for if the cross section has been added/written before fails, and the same GENBEAM cross section is written to Usfos for ever beam, instead of one unique.
- nodeLoadWriter does not write timeFunctionNodeIDs or isMassList to Usfos. Because the format does not support it. And results shows that it is not needed to send to OOcfem. However, these data should probably be written to Usfos at some point.
- Add Eccentricity to elements such as beams and shell when/if it should be implemented in the usfosinput class in OOCfem. If these eccentricities are controlled by the couplings, ignore this point.
- Today the CfemWrapper.cs are doing calculations with both the Cfem API and by using the file based system. The reason is that it should be easy to see what is the changes to the wrapper, and when the time is come to fully go over to the file based system, and not use the API, it should be easy to find what to remove from the code. For example when calling the API for AddNodeLoad in the wrapper, the next call, in that method, is how the same data is added to the Usfos structure. In that way, it is easy to remove the API call and remain with the functionality of the file based system.
- In the CfemWrapper.cs there are comments at the places where the new system is used such as "//VTU" or "//——...—Write beam elements to Usfos". This is just to clarify where in the wrapper the data is transferred to or from the files from the Focus Konstruksjon's data structure. Since the wrapper class could be difficult to navigate in. Make also sure that all parameters that is subtracted with -1 because of the ++ call, such as the m_PropertySetIdxCounter, should not longer be substracted.
- It has to be decided if an analysis file should be written together with the Usfos file. The functionality is written in CfemPlugin.cs, controlled with the

boolean variable writeAnalysisFileAsSeperateFile.

  - In UsfosNodeWriter.cs the method isTranslationIdentityMatrix() must be developed to use the reference UMatrix3x3. If the reference is used, the uncommended code can be used instead of today's code.
  - It has been discussed that the following components has to be developed when further developing the file based system: Spring elements (has to be decided what to do with curves), Truss elements (add components to the Usfos documentation before developing a truss writer) and adding strains.Together with the functionality of the add prescribed displacement method.
  - in the method AddNodeMultiple data is written to the Usfos file. However, the method AddNode does not have it. This is because the code seems a bit redundant, and it is suggested to see if these two methods could be merged before implementing UsfosNodes for both methods.

- Future work on OOcfem side:
  - Important: To merge the changes from the projectKristoffer branch, do not use the automatic "git merge". The time spent changing properties from local paths (because of the VTK library) will take longer time then only merging the code in the changed c++ files.
  - Create new analysis entries for the Usfos documentation, UFO. For the six remaining analysis
  - Analysis parsers in input_usfos.cpp is being overwritten in other parts of the software. Now Cfem_main.cpp only does linear analysis as default.
  - Parsing NODELOAD has now functionallity of defining eccentricity. However, the methods has a comment that says: "// No eccentrisity can be defined through this input". This is commented out since it is now possible to set eccentricity through the updates to the format. This has to be looked into.
  - Functionality of writing eccentricity vectors are available in Focus Konstruksjon. Therefore, writing Eccentricity vectors for elements like beam and

  shells could be possible. However, these are often controlled by couplings. This must be planned with the Focus Konstruksjons developing team.

  - cfemmain.cpp: ANALYSIS_LINEAR has now not any dummy masses for nodes and elements, these are read in the usfos_input class. The same dummy mass elements should be removed from the other analysis, when ready.
  - cfemmain.cpp: Has functionality of taking a third input argument, which is the analysis file. This can be uncommended, when ready. Is controlled by the boolean variable analysisFileExists.
  - Develop a result smoother method for linear_alg_old. When this is completed, Focus Konstruksjon can test results up against the new smoothed results and remove the request of the un-smoothed results in CfemWrapper.cs at line 1455.
  - TimeFunctionIDxs for Loads and masses are not added to the Usfos format. It is not clear if this variable should be utilized the same way it is used in the CfemAPI class.

- Test development:
  - Develop tests for checking if multiple wrapper directories have been created when Focus Konstruksjon is initializing more than one wrapper. Tests today checks if more than one result file have been created when running more than one load combination.
  - Test that truncating values are done correctly. Some tests are written for that purpose today, but for more complex constructions, this is important.
  - Obs: Tests does not complete if files that are being tested are opened in other programs, such as notepad.
  - Test if shell loads, shell masses and node masses are written corretly. Escecially when having dynamic analysis.
  - Writing tests for composit elements - is now manually tested to give correct results (with un-smoothed results).

Appendix D

File formats appendix

Pre-processing – Input files

femML – FEM markup language

The file format femML is developed from the same inspiration as STEP AP209. It is addressing the problems of interoperability in data structures in 3D modelling, and is created to contribute to a vertical integration within the finite element domain [6]. This way the format can facilitate the data transfer, exchange, interchange and integration between finite element applications. Compared to AP209 this format is only focusing on pre-processing data.

XML is a markup language. Originally the language was constructed to represent documents in a document-centric category, now it also used in a data-centric way with representation of data structures, often in web services [50]. The femML is based on a XML format and have a document type definition that represents the categories, called FEM file elements in the report. Such as nodes, elements, material models and load cases.

Figure D.1: femML

Research shows that the format is no longer a priority due to little new information in the documentation the last 10 years. The format is not well-documented nor well-defined, femML is open sourced. However, it is not meeting the requirement of interoperability and implementing it could cause a risk since the format could not exist in the future.

Unfortunate these efforts of making a cross FEM application file format are often shot down after a period of time, research discovered. It is a marked need and a business issue these efforts tries to solve. However, the enthusiasm around these types of formats are by now not found in the large commercial software companies. These companies have politics that makes these formats no longer profitable, and stopping the development.

FEMtools – FE data interfaces

After evaluating two formats wanting to generalize FEM data, the report now investigates FEM software which have data structures and software similar to Focus Konstruksjon. Researching these software solutions, it was early revealed that the licenses were a paid solution. However, they are worth evaluating since they provide some solutions that could be interesting for Focus Software.

Figure D.2: FEMtools

FEMtools is a software that provides GUI for 3D modelling. When users want to do simulations of these model, FEMtools provides a variety of interfaces to exchange data with other FE software for analysis. Such as ABAQUS, ANSYS and NASTRAN [51]. FEMtools uses a Universal File(UF) format interface. The UF is a format developed by Structural Dynamics Research Corporation (SDRC) to make a standard data transfer between CAD models and analysis [52]. From this interface, FEMtools have made it possible to communicate with other large commercial software companies. Instead of isolating the software from other competitors, they have made their solution more general, and integrated other common solutions. An API were made which makes it possible to script how the software should utilize different file formats interfaces, like the formats mentioned before. This is a marked strategy the report wants to highlight, since Focus Software does not have this approach today. With having this approach FEMtools enables the possibility for users to utilize third-party visualization programs and third-party solvers.

Aladdin and other free FEA software

Formats evaluated in this chapter represents well-defined and well-documented formats. On the other side, there are formats that could be useful for Focus Konstruksjon, although they are all specialized on one specific area. Such as high velocity and deformation, fluids, magnetic fields and biological representations [53]. To use these file formats, they would have to be developed to fit Focus Konstruksjon and OOcfem. There are also formats that are based on simple cases such as 2D or 3D linear analysis. These formats would need an extension to cover the needs of a file based system for Focus Software. They are often developed by two or three professors from different universities, that solved a special case. The format mentioned are often under-documented.

Aladdin is an example of a software with a format that is described above [54]. The analysis tool is providing more specialized analysis then generally found, and has the same FEM element data structure stored in a file, normally seen in other formats. However, the formats do not meet the well-defined and well-documented requirements set for the file based system.

Post-processing – Output result files

Abaqus ODB

The ODB files represents the output database structure from ABAQUS analysis. These simulation data is stored in a binary filed with the extension .odb. To read the data from the output database, it needs the special documentation from Abaqus. Research in Abaqus's documentation shows that it is difficult to find an open sourced documentation that would match the needs of a file based Focus Konstruksjon system [55].

There are API's for Python and C++ related to the OBD format. However, these are interfaces intended on parsing the output database. Which provide no solution for writing to the format, and writing to a binary format without proper documentation could be difficult. This makes the ODB format not suitable for the file based system [56]. There are a solutions for requesting a .fil format from the STEP routines in Abaqus, which is an ASCII file, which is again only focused on output result database. Even if the ODB is a well-used format and from that perspective need a thorough evaluation. The format needs to use instances of Abaqus when generating result data. This does not meet the requirements of Focus Software.

SIMULIA
Abaqus 2016

Figure D.3: Abaqus - Simulia

Nastran – bulk data file

Nastran stands for NASA Structural Analysis and has been used to solve finite elements problems since the 1960's. After 2001 Nastran releases the source code and from then NASTRAN have been developed by other independently by for example NX Nastran.

The bulk data file format is well-defined and well-documented, and research showed that several different software uses it. NASTRAN also have other file formats such as OP2. However, the bulk format is used in majority. This would normally lead to numerous of different parsers and writers for this format. Research found that the claim is true, anyhow these interfaces are developed in to different licensed software, and not open sourced. The format has documentation on how the FEM file format elements, called NASTRAN cards, are built up. However, there are few open sourced interfaces or wrappers involving NASTRAN bulk files.

One relevant API made for NASTRAN bulk data files is called pyNastran [57]. This is a Python high-level interface that supports 200 of the most used NASTRAN cards. Even if Python is not a program language used by Focus Konstruksjon, this could be an inspiration for future work. There are other interfaces that focuses on other NASTRAN files such as OP2, which often provide C++ frameworks. Which are not as well-documented as the bulk data file. Therefore, it might not be recommended to implement the OP2 file format.

Since the file format is used by many software programs, implementing the bulk data file will ensure that data from Focus Konstruksjon can be read by other visualization programs. Especially, programs that are not open sourced, and programs that Focus Software's customers already might use.

Appendix E

Doxygen documentation

This is an automatic generated documentation for the file based system. To prevent an excessively documentation, some parts of the documentation is left out from this chapter. The entire documentation can be found in the digital attachments for this thesis, as a web page solution. In the documentation folder, open the file named "index.html" to access the documentation.

Contents

Chapter 1

Namespace Index

1.1 Packages

Here are the packages with brief descriptions (if available):

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

Namespace Documentation

3.1 ParseVTKFormat Namespace Reference

Classes

- class VTKCorrectIDMapping
- class VTKErrorObserver
- class VTKFieldDataReader
- class VTKgetNameSpecificVTKDataArray
- class VTKPointDataReader
- class VTKreader

3.2 WriteUsfosFormat Namespace Reference

Classes

- class UsfosAnalysisInput
- class UsfosBeamCrossSection
- class UsfosCouplingWriter
- class UsfosCrossSectionWriter
- class UsfosEccentricityVector
- class UsfosElement
- class UsfosElementLoads
- class UsfosElementMasses
- class UsfosElementWriter
- class UsfosFileWriter
- class UsfosLinearCouplings
- class UsfosLoadWriter
- class UsfosMassWriter

- class UsfosMaterialWriter
- class UsfosNodeLoads
- class UsfosNodeMasses
- class UsfosNodeWriter
- class UsfosRigitCouplings
- class UsfosUnitVectors

Enumerations

- enum elementDiscription { elementDiscription.BEAM = 28, elementDiscription.TRISHELL = 30 }

3.2.1 Enumeration Type Documentation

3.2.1.1 elementDiscription

```
enum WriteUsfosFormat.elementDiscription  [strong]
```

Enumerator

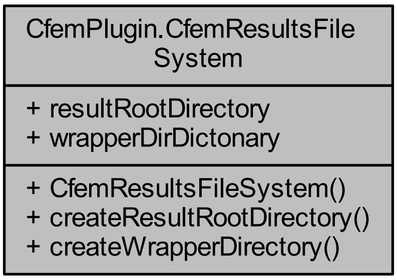
BEAM	
TRISHELL	

Chapter 4

Class Documentation

4.1 CfemPlugin.CfemResultsFileSystem Class Reference

Collaboration diagram for CfemPlugin.CfemResultsFileSystem:


CfemPlugin.CfemResultsFile System
+ resultRootDirectory
+ wrapperDirDictonary
+ CfemResultsFileSystem()
+ createResultRootDirectory()
+ createWrapperDirectory()

Public Member Functions

- CfemResultsFileSystem (string applicationDirectory, string resultDirectory)
- void createResultRootDirectory ()
- void createWrapperDirectory (UID wrapperID)

Properties

- string resultRootDirectory `[get, set]`
- Dictionary$<$ UID, string $>$ wrapperDirDictonary `[get, set]`

4.1.1 Constructor & Destructor Documentation

4.1.1.1 CfemResultsFileSystem()

```
CfemPlugin.CfemResultsFileSystem.CfemResultsFileSystem (
             string applicationDirectory,
             string resultDirectory )
```

4.1.2 Member Function Documentation

4.1.2.1 createResultRootDirectory()

```
void CfemPlugin.CfemResultsFileSystem.createResultRootDirectory ( )
```

4.1.2.2 createWrapperDirectory()

```
void CfemPlugin.CfemResultsFileSystem.createWrapperDirectory (
             UID wrapperID )
```

4.1.3 Property Documentation

4.1.3.1 resultRootDirectory

```
string CfemPlugin.CfemResultsFileSystem.resultRootDirectory  [get], [set]
```

4.1.3.2 wrapperDirDictonary

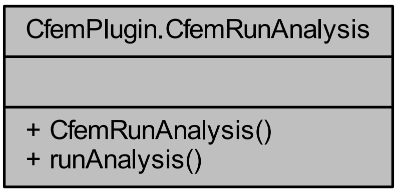
```
Dictionary<UID, string> CfemPlugin.CfemResultsFileSystem.wrapperDirDictonary
[get], [set]
```

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

- CfemResultsFileSystem.cs

4.2 CfemPlugin.CfemRunAnalysis Class Reference

Collaboration diagram for CfemPlugin.CfemRunAnalysis:



Public Member Functions

- CfemRunAnalysis (string workingDirectory, string inputFilePath, string resultWritingDirectory)
- void runAnalysis ()

4.2.1 Constructor & Destructor Documentation

4.2.1.1 CfemRunAnalysis()

```
CfemPlugin.CfemRunAnalysis.CfemRunAnalysis (
             string workingDirectory,
             string inputFilePath,
             string resultWritingDirectory )
```

4.2.2 Member Function Documentation

4.2.2.1 runAnalysis()

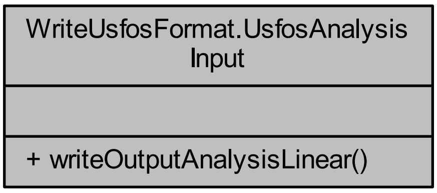
```
void CfemPlugin.CfemRunAnalysis.runAnalysis ( )
```

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

- CfemRunAnalysis.cs

4.3 WriteUsfosFormat.UsfosAnalysisInput Class Reference

Collaboration diagram for WriteUsfosFormat.UsfosAnalysisInput:



Public Member Functions

- List< string > writeOutputAnalysisLinear (CfemInputLinear inputLinear, int outOfPlaneDirection2D)

4.3.1 Member Function Documentation

4.3.1.1 writeOutputAnalysisLinear()

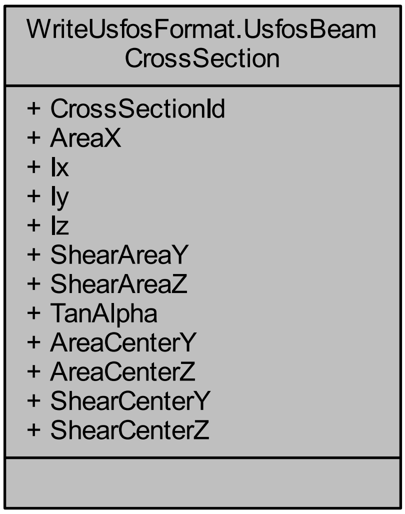
```
List<string> WriteUsfosFormat.UsfosAnalysisInput.writeOutputAnalysisLinear
(
             CfemInputLinear inputLinear,
             int outOfPlaneDirection2D )
```

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

- UsfosAnalysisInput.cs

4.4 WriteUsfosFormat.UsfosBeamCrossSection Class Reference

Collaboration diagram for WriteUsfosFormat.UsfosBeamCrossSection:



Properties

- int CrossSectionId `[get, set]`
- double AreaX `[get, set]`
- double Ix `[get, set]`
- double Iy `[get, set]`
- double Iz `[get, set]`
- double ShearAreaY `[get, set]`
- double ShearAreaZ `[get, set]`
- double TanAlpha `[get, set]`
- double AreaCenterY `[get, set]`
- double AreaCenterZ `[get, set]`
- double ShearCenterY `[get, set]`
- double ShearCenterZ `[get, set]`

4.4.1 Property Documentation

4.4.1.1 AreaCenterY

```
double WriteUsfosFormat.UsfosBeamCrossSection.AreaCenterY  [get], [set]
```

4.4.1.2 AreaCenterZ

```
double WriteUsfosFormat.UsfosBeamCrossSection.AreaCenterZ  [get], [set]
```

4.4.1.3 AreaX

```
double WriteUsfosFormat.UsfosBeamCrossSection.AreaX  [get], [set]
```

4.4.1.4 CrossSectionId

```
int WriteUsfosFormat.UsfosBeamCrossSection.CrossSectionId  [get], [set]
```

4.4.1.5 Ix

```
double WriteUsfosFormat.UsfosBeamCrossSection.Ix  [get], [set]
```

4.4.1.6 Iy

```
double WriteUsfosFormat.UsfosBeamCrossSection.Iy  [get], [set]
```

4.4.1.7 Iz

```
double WriteUsfosFormat.UsfosBeamCrossSection.Iz  [get], [set]
```

4.4.1.8 ShearAreaY

```
double WriteUsfosFormat.UsfosBeamCrossSection.ShearAreaY  [get], [set]
```

4.4.1.9 ShearAreaZ

```
double WriteUsfosFormat.UsfosBeamCrossSection.ShearAreaZ  [get], [set]
```

4.4.1.10 ShearCenterY

```
double WriteUsfosFormat.UsfosBeamCrossSection.ShearCenterY  [get], [set]
```

4.4.1.11 ShearCenterZ

```
double WriteUsfosFormat.UsfosBeamCrossSection.ShearCenterZ  [get], [set]
```

4.4.1.12 TanAlpha

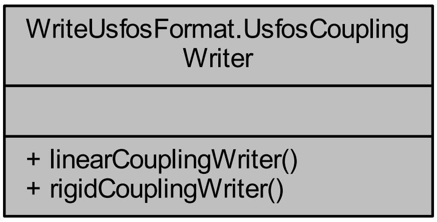
```
double WriteUsfosFormat.UsfosBeamCrossSection.TanAlpha  [get], [set]
```

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

- UsfosBeamCrossSection.cs

4.5 WriteUsfosFormat.UsfosCouplingWriter Class Reference

Collaboration diagram for WriteUsfosFormat.UsfosCouplingWriter:



Public Member Functions

- List< string > linearCouplingWriter (UsfosLinearCouplings linCouplings)
- List< string > rigidCouplingWriter (UsfosRigitCouplings rigidCouplings)

4.5.1 Member Function Documentation

4.5.1.1 linearCouplingWriter()

```
List<string> WriteUsfosFormat.UsfosCouplingWriter.linearCouplingWriter (
             UsfosLinearCouplings linCouplings )
```

4.5.1.2 rigidCouplingWriter()

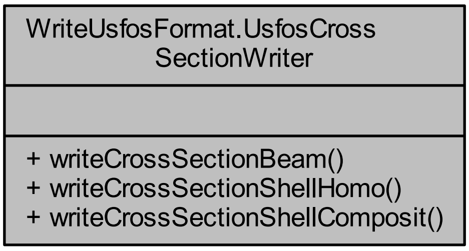
```
List<string> WriteUsfosFormat.UsfosCouplingWriter.rigidCouplingWriter (
             UsfosRigitCouplings rigidCouplings )
```

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

- UsfosCouplingWriter.cs

4.6 WriteUsfosFormat.UsfosCrossSectionWriter Class Reference

Collaboration diagram for WriteUsfosFormat.UsfosCrossSectionWriter:


WriteUsfosFormat.UsfosCross SectionWriter
+ writeCrossSectionBeam()
+ writeCrossSectionShellHomo()
+ writeCrossSectionShellComposit()

Public Member Functions

- List< string > writeCrossSectionBeam (UsfosBeamCrossSection bProp)
- List< string > writeCrossSectionShellHomo (int CrossSectionId, double thickness)
- string writeCrossSectionShellComposit (int CompositSectionId, double z0, int[ ] materialIDs, double[ ] thicknessList, double[ ] thetalOrientationList)

4.6.1 Member Function Documentation

4.6.1.1 writeCrossSectionBeam()

```
List<string> WriteUsfosFormat.UsfosCrossSectionWriter.writeCrossSectionBeam
(
            UsfosBeamCrossSection bProp )
```

4.6.1.2 writeCrossSectionShellComposit()

```
string WriteUsfosFormat.UsfosCrossSectionWriter.writeCrossSectionShellComposit
(
            int CompositSectionId,
            double z0,
            int [] materialIDs,
            double [] thicknessList,
            double [] thetalOrientationList )
```

4.6.1.3 writeCrossSectionShellHomo()

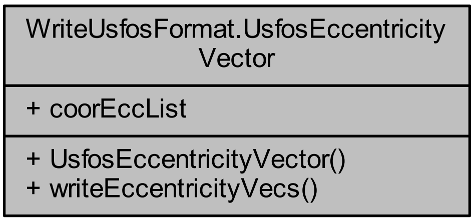
```
List<string> WriteUsfosFormat.UsfosCrossSectionWriter.writeCrossSectionShell←
Homo (
            int CrossSectionId,
            double thickness )
```

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

- UsfosCrossSectionWriter.cs

4.7 WriteUsfosFormat.UsfosEccentricityVector Class Reference

Collaboration diagram for WriteUsfosFormat.UsfosEccentricityVector:


WriteUsfosFormat.UsfosEccentricity
Vector
+ coorEccList
+ UsfosEccentricityVector()
+ writeEccentricityVecs()

Public Member Functions

- UsfosEccentricityVector ()
- List< string > writeEccentricityVecs ()

Properties

- List< double[ ]> coorEccList `[get, set]`

4.7.1 Constructor & Destructor Documentation

4.7.1.1 UsfosEccentricityVector()

```
WriteUsfosFormat.UsfosEccentricityVector.UsfosEccentricityVector ( )
```

4.7.2 Member Function Documentation

4.7.2.1 writeEccentricityVecs()

```
List<string> WriteUsfosFormat.UsfosEccentricityVector.writeEccentricityVecs
( )
```

4.7.3 Property Documentation

4.7.3.1 coorEccList

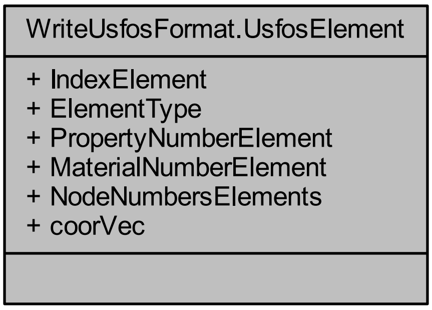
```
List<double[ ]> WriteUsfosFormat.UsfosEccentricityVector.coorEccList [get],
[set]
```

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

- UsfosEccentricityVector.cs

4.8 WriteUsfosFormat.UsfosElement Class Reference

Collaboration diagram for WriteUsfosFormat.UsfosElement:



Properties

- int [ ] IndexElement `[get, set]`
- int [ ] ElementType `[get, set]`
- int [ ] PropertyNumberElement `[get, set]`
- int [ ] MaterialNumberElement `[get, set]`
- int [,] NodeNumbersElements `[get, set]`
- double [ ][ ] coorVec `[get, set]`

4.8.1 Property Documentation

4.8.1.1 coorVec

```
double [][] WriteUsfosFormat.UsfosElement.coorVec [get], [set]
```

4.8.1.2 ElementType

```
int [] WriteUsfosFormat.UsfosElement.ElementType [get], [set]
```

4.8.1.3 IndexElement

```
int [] WriteUsfosFormat.UsfosElement.IndexElement  [get], [set]
```

4.8.1.4 MaterialNumberElement

```
int [] WriteUsfosFormat.UsfosElement.MaterialNumberElement  [get], [set]
```

4.8.1.5 NodeNumbersElements

```
int [,] WriteUsfosFormat.UsfosElement.NodeNumbersElements  [get], [set]
```

4.8.1.6 PropertyNumberElement

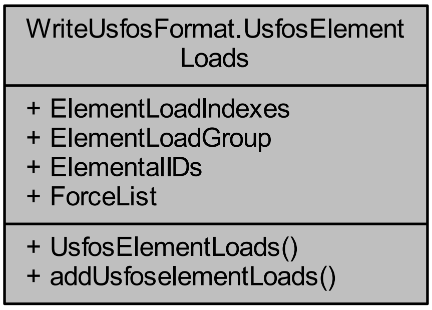
```
int [] WriteUsfosFormat.UsfosElement.PropertyNumberElement  [get], [set]
```

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

- UsfosElement.cs

4.9 WriteUsfosFormat.UsfosElementLoads Class Reference

Collaboration diagram for WriteUsfosFormat.UsfosElementLoads:



Public Member Functions

- UsfosElementLoads ()
- void addUsfoselementLoads (int loadIndex, int loadGroup, int[ ] elementalIDs, double[ ] forces)

Properties

- List< int > ElementLoadIndexes `[get, set]`
- List< int > ElementLoadGroup `[get, set]`
- List< int[ ]> ElementalIDs `[get, set]`
- List< double[ ]> ForceList `[get, set]`

4.9.1 Constructor & Destructor Documentation

4.9.1.1 UsfosElementLoads()

```
WriteUsfosFormat.UsfosElementLoads.UsfosElementLoads ( )
```

4.9.2 Member Function Documentation

4.9.2.1 addUsfoselementLoads()

```
void WriteUsfosFormat.UsfosElementLoads.addUsfoselementLoads (
            int loadIndex,
            int loadGroup,
            int [ ] elementalIDs,
            double [ ] forces )
```

4.9.3 Property Documentation

4.9.3.1 ElementalIDs

```
List<int[ ]> WriteUsfosFormat.UsfosElementLoads.ElementalIDs [get], [set]
```

4.9.3.2 ElementLoadGroup

```
List<int> WriteUsfosFormat.UsfosElementLoads.ElementLoadGroup [get], [set]
```

4.9.3.3 ElementLoadIndexes

```
List<int> WriteUsfosFormat.UsfosElementLoads.ElementLoadIndexes  [get], [set]
```

4.9.3.4 ForceList

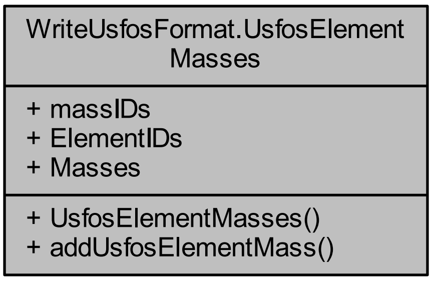
```
List<double[]> WriteUsfosFormat.UsfosElementLoads.ForceList  [get], [set]
```

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

- UsfosElementLoads.cs

4.10 WriteUsfosFormat.UsfosElementMasses Class Reference

Collaboration diagram for WriteUsfosFormat.UsfosElementMasses:



Public Member Functions

- UsfosElementMasses ()
- void addUsfosElementMass (int massID, int[ ] elementIDs, double mass)

Properties

- List< int > massIDs `[get, set]`
- List< int[ ]> ElementIDs `[get, set]`
- List< double > Masses `[get, set]`

4.10.1 Constructor & Destructor Documentation

4.10.1.1 UsfosElementMasses()

```
WriteUsfosFormat.UsfosElementMasses.UsfosElementMasses ( )
```

4.10.2 Member Function Documentation

4.10.2.1 addUsfosElementMass()

```
void WriteUsfosFormat.UsfosElementMasses.addUsfosElementMass (
            int massID,
            int [] elementIDs,
            double mass )
```

4.10.3 Property Documentation

4.10.3.1 ElementIDs

```
List<int[]> WriteUsfosFormat.UsfosElementMasses.ElementIDs [get], [set]
```

4.10.3.2 Masses

```
List<double> WriteUsfosFormat.UsfosElementMasses.Masses [get], [set]
```

4.10.3.3 massIDs

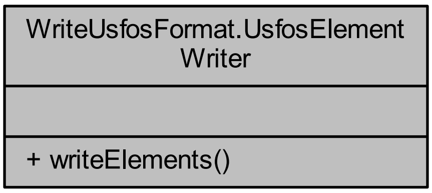
```
List<int> WriteUsfosFormat.UsfosElementMasses.massIDs [get], [set]
```

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

- UsfosElementMasses.cs

4.11 WriteUsfosFormat.UsfosElementWriter Class Reference

Collaboration diagram for WriteUsfosFormat.UsfosElementWriter:



Public Member Functions

- List< string > writeElements (UsfosElement usfosElement, UsfosUnitVectors unitVecs)

4.11.1 Member Function Documentation

4.11.1.1 writeElements()

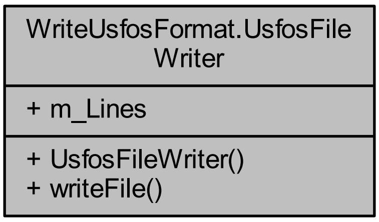
```
List<string> WriteUsfosFormat.UsfosElementWriter.writeElements (
             UsfosElement usfosElement,
             UsfosUnitVectors unitVecs )
```

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

- UsfosElementWriter.cs

4.12 WriteUsfosFormat.UsfosFileWriter Class Reference

Collaboration diagram for WriteUsfosFormat.UsfosFileWriter:



Public Member Functions

- UsfosFileWriter (string filePath)
- void writeFile ()

Properties

- List< string > m_Lines `[get, set]`

4.12.1 Constructor & Destructor Documentation

4.12.1.1 UsfosFileWriter()

```
WriteUsfosFormat.UsfosFileWriter.UsfosFileWriter (
             string filePath )
```

4.12.2 Member Function Documentation

4.12.2.1 writeFile()

```
void WriteUsfosFormat.UsfosFileWriter.writeFile ( )
```

4.12.3 Property Documentation

4.12.3.1 m_Lines

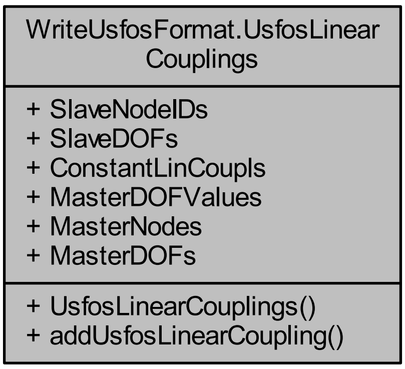
`List<string> WriteUsfosFormat.UsfosFileWriter.m_Lines [get], [set]`

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

- UsforsFileWriter.cs

4.13 WriteUsfosFormat.UsfosLinearCouplings Class Reference

Collaboration diagram for WriteUsfosFormat.UsfosLinearCouplings:



Public Member Functions

- UsfosLinearCouplings ()
- void addUsfosLinearCoupling (int slaveNodeID, int slaveDOF, double constLinCoupl, double[ ] masterDOFvalue, int[ ] masterNodes, int[ ] masterDOF)

Properties

- List< int > SlaveNodeIDs `[get, set]`
- List< int > SlaveDOFs `[get, set]`
- List< double > ConstantLinCoupls `[get, set]`
- List< double[ ]> MasterDOFValues `[get, set]`
- List< int[ ]> MasterNodes `[get, set]`
- List< int[ ]> MasterDOFs `[get, set]`

4.13.1 Constructor & Destructor Documentation

4.13.1.1 UsfosLinearCouplings()

```
WriteUsfosFormat.UsfosLinearCouplings.UsfosLinearCouplings ( )
```

4.13.2 Member Function Documentation

4.13.2.1 addUsfosLinearCoupling()

```
void WriteUsfosFormat.UsfosLinearCouplings.addUsfosLinearCoupling (
             int slaveNodeID,
             int slaveDOF,
             double constLinCoupl,
             double [] masterDOFvalue,
             int [] masterNodes,
             int [] masterDOF )
```

4.13.3 Property Documentation

4.13.3.1 ConstantLinCoupls

```
List<double> WriteUsfosFormat.UsfosLinearCouplings.ConstantLinCoupls  [get],
[set]
```

4.13.3.2 MasterDOFs

```
List<int[]> WriteUsfosFormat.UsfosLinearCouplings.MasterDOFs  [get], [set]
```

4.13.3.3 MasterDOFValues

```
List<double[]> WriteUsfosFormat.UsfosLinearCouplings.MasterDOFValues  [get],
[set]
```

4.13.3.4 MasterNodes

```
List<int[]> WriteUsfosFormat.UsfosLinearCouplings.MasterNodes  [get], [set]
```

4.13.3.5 SlaveDOFs

```
List<int> WriteUsfosFormat.UsfosLinearCouplings.SlaveDOFs  [get], [set]
```

4.13.3.6 SlaveNodeIDs

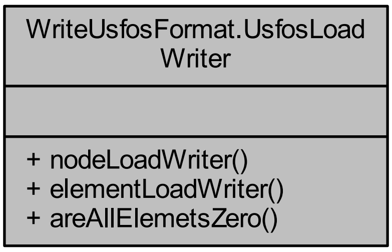
```
List<int> WriteUsfosFormat.UsfosLinearCouplings.SlaveNodeIDs  [get], [set]
```

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

- UsfosLinearCouplings.cs

4.14 WriteUsfosFormat.UsfosLoadWriter Class Reference

Collaboration diagram for WriteUsfosFormat.UsfosLoadWriter:



Public Member Functions

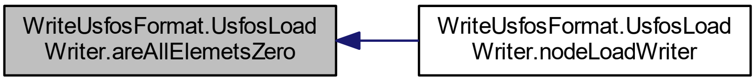
- List< string > nodeLoadWriter (UsfosNodeLoads loads, UsfosEccentricityVector eccVector)
- List< string > elementLoadWriter (UsfosElementLoads elementLoads)
- bool areAllElemetsZero (double[ ] array)

4.14.1 Member Function Documentation

4.14.1.1 areAllElemetsZero()

```
bool WriteUsfosFormat.UsfosLoadWriter.areAllElemetsZero (
             double [ ] array )
```

Here is the caller graph for this function:


WriteUsfosFormat.UsfosLoad Writer.areAllElemetsZero
WriteUsfosFormat.UsfosLoad Writer.nodeLoadWriter

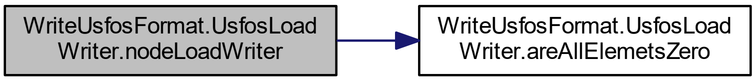
4.14.1.2 elementLoadWriter()

```
List<string> WriteUsfosFormat.UsfosLoadWriter.elementLoadWriter (
             UsfosElementLoads elementLoads )
```

4.14.1.3 nodeLoadWriter()

```
List<string> WriteUsfosFormat.UsfosLoadWriter.nodeLoadWriter (
             UsfosNodeLoads loads,
             UsfosEccentricityVector eccVector )
```

Here is the call graph for this function:

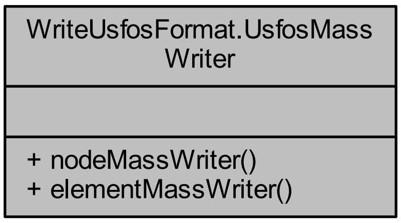

WriteUsfosFormat.UsfosLoad Writer.nodeLoadWriter
WriteUsfosFormat.UsfosLoad Writer.areAllElemetsZero

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

- UsfosLoadWriter.cs

4.15 WriteUsfosFormat.UsfosMassWriter Class Reference

Collaboration diagram for WriteUsfosFormat.UsfosMassWriter:



Public Member Functions

- List< string > nodeMassWriter (UsfosNodeMasses nodeMass)
- List< string > elementMassWriter (UsfosElementMasses elementMasses)

4.15.1 Member Function Documentation

4.15.1.1 elementMassWriter()

```
List<string> WriteUsfosFormat.UsfosMassWriter.elementMassWriter (
             UsfosElementMasses elementMasses )
```

4.15.1.2 nodeMassWriter()

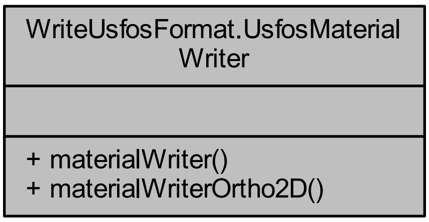
```
List<string> WriteUsfosFormat.UsfosMassWriter.nodeMassWriter (
             UsfosNodeMasses nodeMass )
```

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

- UsfosMassWriter.cs

4.16 WriteUsfosFormat.UsfosMaterialWriter Class Reference

Collaboration diagram for WriteUsfosFormat.UsfosMaterialWriter:


WriteUsfosFormat.UsfosMaterial Writer
+ materialWriter()
+ materialWriterOrtho2D()

Public Member Functions

- string materialWriter (int materialId, double materialEmod, double materialPoissNu, double Material↩ DensityRho)
- string materialWriterOrtho2D (int materialId, double Ex1, double Ey2, double materialPoissNu, double G12, double MaterialDensityRho)

4.16.1 Member Function Documentation

4.16.1.1 materialWriter()

```
string WriteUsfosFormat.UsfosMaterialWriter.materialWriter (
             int materialId,
             double materialEmod,
             double materialPoissNu,
             double MaterialDensityRho )
```

4.16.1.2 materialWriterOrtho2D()

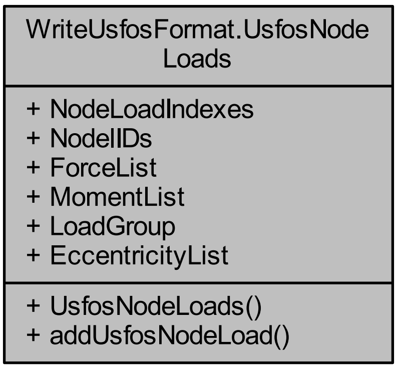
```
string WriteUsfosFormat.UsfosMaterialWriter.materialWriterOrtho2D (
             int materialId,
             double Ex1,
             double Ey2,
             double materialPoissNu,
             double G12,
             double MaterialDensityRho )
```

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

- UsfosMaterialWriter.cs

4.17 WriteUsfosFormat.UsfosNodeLoads Class Reference

Collaboration diagram for WriteUsfosFormat.UsfosNodeLoads:



Public Member Functions

- UsfosNodeLoads ()
- void addUsfosNodeLoad (int nodeLoadIndexes, int nodeIID, double[ ] forceList, double[ ] moment↩ List, int loadGroup, double[ ] eccVector)

Properties

- List< int > NodeLoadIndexes `[get, set]`
- List< int > NodeIIDs `[get, set]`
- List< double[ ]> ForceList `[get, set]`
- List< double[ ]> MomentList `[get, set]`
- List< int > LoadGroup `[get, set]`
- List< double[ ]> EccentricityList `[get, set]`

4.17.1 Constructor & Destructor Documentation

4.17.1.1 UsfosNodeLoads()

```
WriteUsfosFormat.UsfosNodeLoads.UsfosNodeLoads ( )
```

4.17.2 Member Function Documentation

4.17.2.1 addUsfosNodeLoad()

```
void WriteUsfosFormat.UsfosNodeLoads.addUsfosNodeLoad (
            int nodeLoadIndexes,
            int nodeIID,
            double [] forceList,
            double [] momentList,
            int loadGroup,
            double [] eccVector )
```

4.17.3 Property Documentation

4.17.3.1 EccentricityList

```
List<double[]> WriteUsfosFormat.UsfosNodeLoads.EccentricityList [get], [set]
```

4.17.3.2 ForceList

```
List<double[]> WriteUsfosFormat.UsfosNodeLoads.ForceList  [get], [set]
```

4.17.3.3 LoadGroup

```
List<int> WriteUsfosFormat.UsfosNodeLoads.LoadGroup  [get], [set]
```

4.17.3.4 MomentList

```
List<double[]> WriteUsfosFormat.UsfosNodeLoads.MomentList  [get], [set]
```

4.17.3.5 NodelIDs

```
List<int> WriteUsfosFormat.UsfosNodeLoads.NodelIDs  [get], [set]
```

4.17.3.6 NodeLoadIndexes

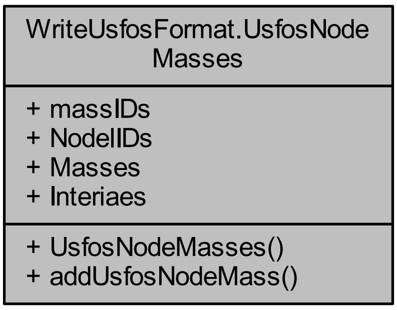
```
List<int> WriteUsfosFormat.UsfosNodeLoads.NodeLoadIndexes  [get], [set]
```

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

- UsfosNodeLoads.cs

4.18 WriteUsfosFormat.UsfosNodeMasses Class Reference

Collaboration diagram for WriteUsfosFormat.UsfosNodeMasses:



Public Member Functions

- UsfosNodeMasses ()
- void addUsfosNodeMass (int massID, int nodelID, double mass, double interia)

Properties

- List< int > massIDs `[get, set]`
- List< int > NodelIDs `[get, set]`
- List< double > Masses `[get, set]`
- List< double > Interiaes `[get, set]`

4.18.1 Constructor & Destructor Documentation

4.18.1.1 UsfosNodeMasses()

```
WriteUsfosFormat.UsfosNodeMasses.UsfosNodeMasses ( )
```

4.18.2 Member Function Documentation

4.18.2.1 addUsfosNodeMass()

```
void WriteUsfosFormat.UsfosNodeMasses.addUsfosNodeMass (
            int massID,
            int nodelID,
            double mass,
            double interia )
```

4.18.3 Property Documentation

4.18.3.1 Interiaes

```
List<double> WriteUsfosFormat.UsfosNodeMasses.Interiaes [get], [set]
```

4.18.3.2 Masses

```
List<double> WriteUsfosFormat.UsfosNodeMasses.Masses [get], [set]
```

4.18.3.3 massIDs

`List<int> WriteUsfosFormat.UsfosNodeMasses.massIDs [get], [set]`

4.18.3.4 NodelIDs

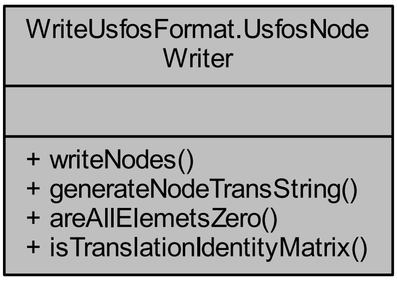
`List<int> WriteUsfosFormat.UsfosNodeMasses.NodelIDs [get], [set]`

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

- UsfosNodeMasses.cs

4.19 WriteUsfosFormat.UsfosNodeWriter Class Reference

Collaboration diagram for WriteUsfosFormat.UsfosNodeWriter:


WriteUsfosFormat.UsfosNode Writer
+ writeNodes()
+ generateNodeTransString()
+ areAllElemetsZero()
+ isTranslationIdentityMatrix()

Public Member Functions

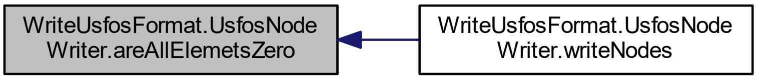
- List< string > writeNodes (int[ ] nodeIdexes, double[ ][ ] nodeXYZ, int[ ][ ] bCode, double[ ][,] trans)
- string generateNodeTransString (int numberOfNodeTrans, double[,] trans)
- bool areAllElemetsZero (int[ ] array)
- bool isTranslationIdentityMatrix (double[,] transMatrix)

4.19.1 Member Function Documentation

4.19.1.1 areAllElemetsZero()

```
bool WriteUsfosFormat.UsfosNodeWriter.areAllElemetsZero (
            int [] array )
```

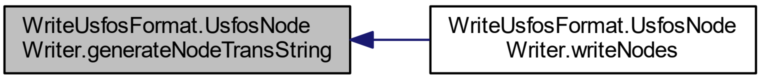
Here is the caller graph for this function:



4.19.1.2 generateNodeTransString()

```
string WriteUsfosFormat.UsfosNodeWriter.generateNodeTransString (
            int numberOfNodeTrans,
            double trans[,] )
```

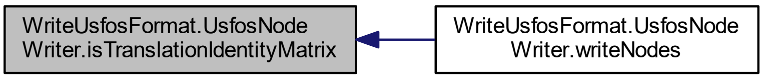
Here is the caller graph for this function:


WriteUsfosFormat.UsfosNode
Writer.generateNodeTransString
WriteUsfosFormat.UsfosNode
Writer.writeNodes

4.19.1.3 isTranslationIdentityMatrix()

```
bool WriteUsfosFormat.UsfosNodeWriter.isTranslationIdentityMatrix (
            double transMatrix[,] )
```

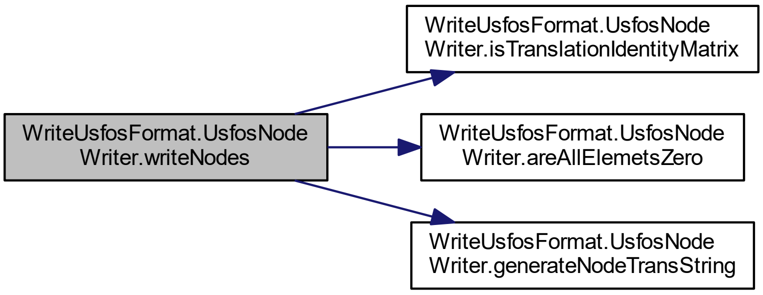
Here is the caller graph for this function:


WriteUsfosFormat.UsfosNode Writer.isTranslationIdentityMatrix
WriteUsfosFormat.UsfosNode Writer.writeNodes

4.19.1.4 writeNodes()

```
List<string> WriteUsfosFormat.UsfosNodeWriter.writeNodes (
              int [] nodeIdexes,
              double nodeXYZ[][],
              int bCode[][],
              double trans[][,] )
```

Here is the call graph for this function:

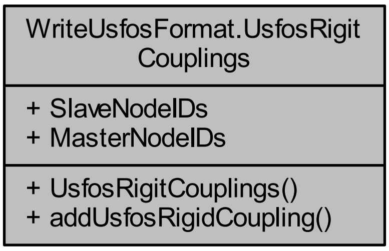

WriteUsfosFormat.UsfosNode Writer.writeNodes
WriteUsfosFormat.UsfosNode Writer.isTranslationIdentityMatrix
WriteUsfosFormat.UsfosNode Writer.areAllElemetsZero
WriteUsfosFormat.UsfosNode Writer.generateNodeTransString

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

- UsfosNodeWriter.cs

4.20 WriteUsfosFormat.UsfosRigitCouplings Class Reference

Collaboration diagram for WriteUsfosFormat.UsfosRigitCouplings:


WriteUsfosFormat.UsfosRigit Couplings
+ SlaveNodeIDs
+ MasterNodeIDs
+ UsfosRigitCouplings()
+ addUsfosRigidCoupling()

Public Member Functions

- UsfosRigitCouplings ()
- void addUsfosRigidCoupling (int slaveNodeID, int masterNodeID)

Properties

- List< int > SlaveNodeIDs `[get, set]`
- List< int > MasterNodeIDs `[get, set]`

4.20.1 Constructor & Destructor Documentation

4.20.1.1 UsfosRigitCouplings()

```
WriteUsfosFormat.UsfosRigitCouplings.UsfosRigitCouplings ( )
```

4.20.2 Member Function Documentation

4.20.2.1 addUsfosRigidCoupling()

```
void WriteUsfosFormat.UsfosRigitCouplings.addUsfosRigidCoupling (
             int slaveNodeID,
             int masterNodeID )
```

4.20.3 Property Documentation

4.20.3.1 MasterNodeIDs

```
List<int> WriteUsfosFormat.UsfosRigitCouplings.MasterNodeIDs  [get], [set]
```

4.20.3.2 SlaveNodeIDs

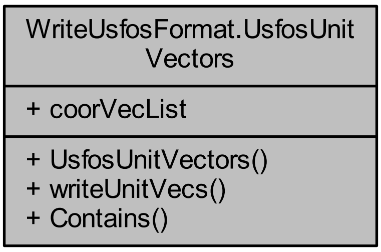
```
List<int> WriteUsfosFormat.UsfosRigitCouplings.SlaveNodeIDs  [get], [set]
```

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

- UsfosRigitCouplings.cs

4.21 WriteUsfosFormat.UsfosUnitVectors Class Reference

Collaboration diagram for WriteUsfosFormat.UsfosUnitVectors:



Public Member Functions

- UsfosUnitVectors ()
- List< string > writeUnitVecs ()
- int Contains (double[ ] toBeChecked)

Properties

- List< double[ ]> coorVecList `[get, set]`

4.21.1 Constructor & Destructor Documentation

4.21.1.1 UsfosUnitVectors()

```
WriteUsfosFormat.UsfosUnitVectors.UsfosUnitVectors ( )
```

4.21.2 Member Function Documentation

4.21.2.1 Contains()

```
int WriteUsfosFormat.UsfosUnitVectors.Contains (
             double [] toBeChecked )
```

4.21.2.2 writeUnitVecs()

```
List<string> WriteUsfosFormat.UsfosUnitVectors.writeUnitVecs ( )
```

4.21.3 Property Documentation

4.21.3.1 coorVecList

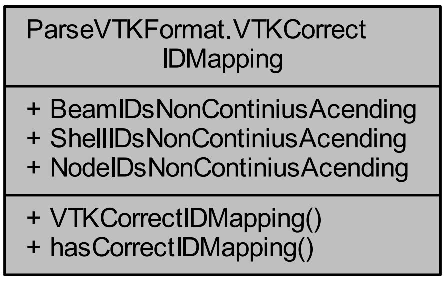
```
List<double[]> WriteUsfosFormat.UsfosUnitVectors.coorVecList  [get], [set]
```

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

- UsfosUnitVectors.cs

4.22 ParseVTKFormat.VTKCorrectIDMapping Class Reference

Collaboration diagram for ParseVTKFormat.VTKCorrectIDMapping:


ParseVTKFormat.VTKCorrect
IDMapping
+ BeamIDsNonContiniusAcending
+ ShellIDsNonContiniusAcending
+ NodeIDsNonContiniusAcending
+ VTKCorrectIDMapping()
+ hasCorrectIDMapping()

Public Member Functions

- VTKCorrectIDMapping (vtkUnstructuredGrid unstructuredGrid)
- bool hasCorrectIDMapping (string idType, int[ ] idList)

Properties

- int [ ] BeamIDsNonContiniusAcending `[get]`
- int [ ] ShellIDsNonContiniusAcending `[get]`
- int [ ] NodeIDsNonContiniusAcending `[get]`

4.22.1 Constructor & Destructor Documentation

4.22.1.1 VTKCorrectIDMapping()

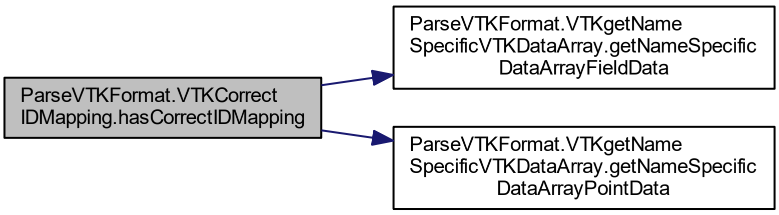
```
ParseVTKFormat.VTKCorrectIDMapping.VTKCorrectIDMapping (
            vtkUnstructuredGrid unstructuredGrid )
```

4.22.2 Member Function Documentation

4.22.2.1 hasCorrectIDMapping()

```
bool ParseVTKFormat.VTKCorrectIDMapping.hasCorrectIDMapping (
             string idType,
             int [] idList )
```

Here is the call graph for this function:


ParseVTKFormat.VTKCorrect IDMapping.hasCorrectIDMapping
ParseVTKFormat.VTKgetName SpecificVTKDataArray.getNameSpecific DataArrayFieldData
ParseVTKFormat.VTKgetName SpecificVTKDataArray.getNameSpecific DataArrayPointData

4.22.3 Property Documentation

4.22.3.1 BeamIDsNonContiniusAcending

```
int [] ParseVTKFormat.VTKCorrectIDMapping.BeamIDsNonContiniusAcending  [get]
```

4.22.3.2 NodeIDsNonContiniusAcending

```
int [] ParseVTKFormat.VTKCorrectIDMapping.NodeIDsNonContiniusAcending  [get]
```

4.22.3.3 ShellIDsNonContiniusAcending

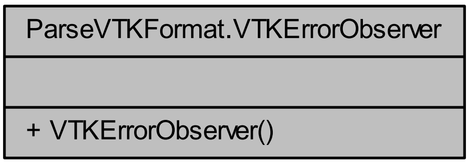
```
int [] ParseVTKFormat.VTKCorrectIDMapping.ShellIDsNonContiniusAcending  [get]
```

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

- VTKCorrectIDMapping.cs

4.23 ParseVTKFormat.VTKErrorObserver Class Reference

Collaboration diagram for ParseVTKFormat.VTKErrorObserver:



Public Member Functions

- VTKErrorObserver ()

4.23.1 Constructor & Destructor Documentation

4.23.1.1 VTKErrorObserver()

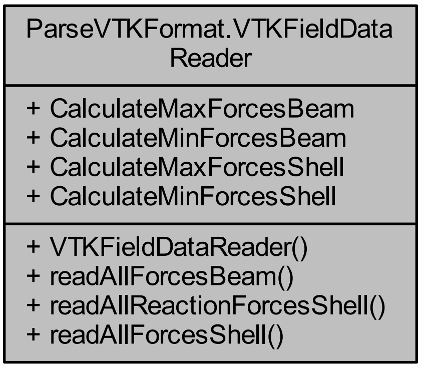
```
ParseVTKFormat.VTKErrorObserver.VTKErrorObserver ( )
```

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

- VTKErrorObserver.cs

4.24 ParseVTKFormat.VTKFieldDataReader Class Reference

Collaboration diagram for ParseVTKFormat.VTKFieldDataReader:


ParseVTKFormat.VTKFieldData Reader
+ CalculateMaxForcesBeam
+ CalculateMinForcesBeam
+ CalculateMaxForcesShell
+ CalculateMinForcesShell
+ VTKFieldDataReader()
+ readAllForcesBeam()
+ readAllReactionForcesShell()
+ readAllForcesShell()

Public Member Functions

- VTKFieldDataReader (vtkUnstructuredGrid unstructuredGrid)
- double [,] readAllForcesBeam (bool hasCorrectIDMapping, int[ ] correctedElID)
- double [,] readAllReactionForcesShell (bool hasCorrectIDMapping, int[ ] correctedElID)
- double [ ][ ] readAllForcesShell (bool hasCorrectIDMapping, int[ ] correctedElID)

Properties

- double [ ] CalculateMaxForcesBeam `[get]`
- double [ ] CalculateMinForcesBeam `[get]`
- double [ ] CalculateMaxForcesShell `[get]`
- double [ ] CalculateMinForcesShell `[get]`

4.24.1 Constructor & Destructor Documentation

4.24.1.1 VTKFieldDataReader()

```
ParseVTKFormat.VTKFieldDataReader.VTKFieldDataReader (
             vtkUnstructuredGrid unstructuredGrid )
```

4.24.2 Member Function Documentation

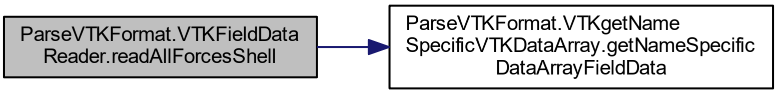
4.24.2.1 readAllForcesBeam()

```
double [,] ParseVTKFormat.VTKFieldDataReader.readAllForcesBeam (
             bool hasCorrectIDMapping,
             int [] correctedElID )
```

4.24.2.2 readAllForcesShell()

```
double [][] ParseVTKFormat.VTKFieldDataReader.readAllForcesShell (
             bool hasCorrectIDMapping,
             int [] correctedElID )
```

Here is the call graph for this function:


ParseVTKFormat.VTKFieldData Reader.readAllForcesShell
ParseVTKFormat.VTKgetName SpecificVTKDataArray.getNameSpecific DataArrayFieldData

4.24.2.3 readAllReactionForcesShell()

```
double [,] ParseVTKFormat.VTKFieldDataReader.readAllReactionForcesShell (
             bool hasCorrectIDMapping,
             int [] correctedElID )
```

4.24.3 Property Documentation

4.24.3.1 CalculateMaxForcesBeam

```
double [] ParseVTKFormat.VTKFieldDataReader.CalculateMaxForcesBeam  [get]
```

4.24.3.2 CalculateMaxForcesShell

```
double [] ParseVTKFormat.VTKFieldDataReader.CalculateMaxForcesShell  [get]
```

4.24.3.3 CalculateMinForcesBeam

```
double [] ParseVTKFormat.VTKFieldDataReader.CalculateMinForcesBeam  [get]
```

4.24.3.4 CalculateMinForcesShell

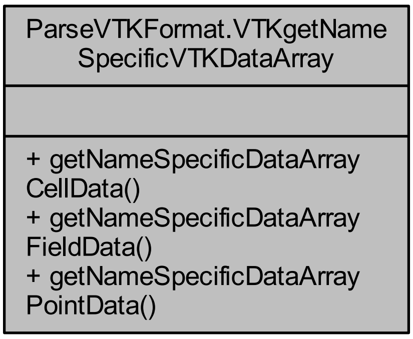
```
double [] ParseVTKFormat.VTKFieldDataReader.CalculateMinForcesShell  [get]
```

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

- VTKFieldDataReader.cs

4.25 ParseVTKFormat.VTKgetNameSpecificVTKDataArray Class Reference

Collaboration diagram for ParseVTKFormat.VTKgetNameSpecificVTKDataArray:


ParseVTKFormat.VTKgetName SpecificVTKDataArray
+ getNameSpecificDataArray CellData()
+ getNameSpecificDataArray FieldData()
+ getNameSpecificDataArray PointData()

Public Member Functions

- vtkDataArray getNameSpecificDataArrayCellData (vtkUnstructuredGrid unstructuredGrid, string arrayName)
- vtkDataArray getNameSpecificDataArrayFieldData (vtkUnstructuredGrid unstructuredGrid, string arrayName)
- vtkDataArray getNameSpecificDataArrayPointData (vtkUnstructuredGrid unstructuredGrid, string arrayName)

4.25.1 Member Function Documentation

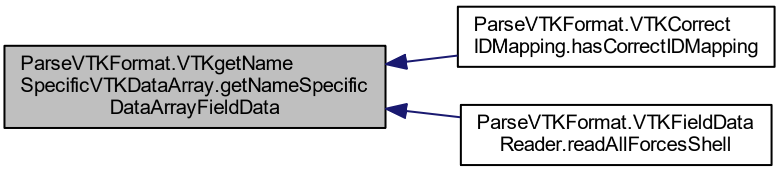
4.25.1.1 getNameSpecificDataArrayCellData()

```
vtkDataArray ParseVTKFormat.VTKgetNameSpecificVTKDataArray.getNameSpecific←↩
DataArrayCellData (
             vtkUnstructuredGrid unstructuredGrid,
             string arrayName )
```

4.25.1.2 getNameSpecificDataArrayFieldData()

```
vtkDataArray ParseVTKFormat.VTKgetNameSpecificVTKDataArray.getNameSpecific←↩
DataArrayFieldData (
             vtkUnstructuredGrid unstructuredGrid,
             string arrayName )
```

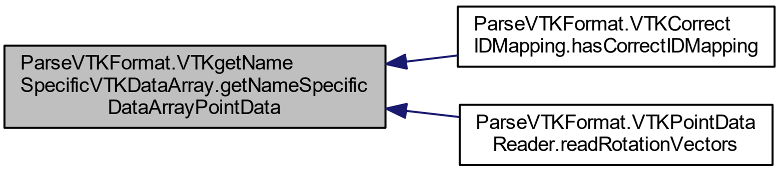
Here is the caller graph for this function:


ParseVTKFormat.VTKgetName SpecificVTKDataArray.getNameSpecific DataArrayFieldData
ParseVTKFormat.VTKCorrect IDMapping.hasCorrectIDMapping
ParseVTKFormat.VTKFieldData Reader.readAllForcesShell

4.25.1.3 getNameSpecificDataArrayPointData()

```
vtkDataArray ParseVTKFormat.VTKgetNameSpecificVTKDataArray.getNameSpecific←↩
DataArrayPointData (
             vtkUnstructuredGrid unstructuredGrid,
             string arrayName )
```

Here is the caller graph for this function:

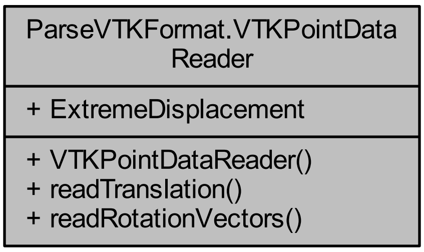

ParseVTKFormat.VTKgetName SpecificVTKDataArray.getNameSpecific DataArrayPointData
ParseVTKFormat.VTKCorrect IDMapping.hasCorrectIDMapping
ParseVTKFormat.VTKPointData Reader.readRotationVectors

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

- VTKgetNameSpecificVTKDataArray.cs

4.26 ParseVTKFormat.VTKPointDataReader Class Reference

Collaboration diagram for ParseVTKFormat.VTKPointDataReader:



Public Member Functions

- VTKPointDataReader (vtkUnstructuredGrid unstructuredGrid)
- double [,] readTranslation (bool hasCorrectIDMapping, int[ ] correctedNodeID)
- double [,] readRotationVectors (bool hasCorrectIDMapping, int[ ] correctedNodeID)

Properties

- double [ ] ExtremeDisplacement `[get]`

4.26.1 Constructor & Destructor Documentation

4.26.1.1 VTKPointDataReader()

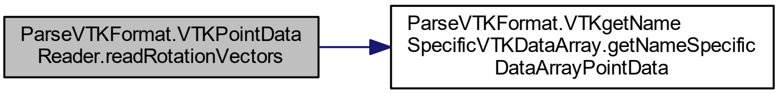
```
ParseVTKFormat.VTKPointDataReader.VTKPointDataReader (
             vtkUnstructuredGrid unstructuredGrid )
```

4.26.2 Member Function Documentation

4.26.2.1 readRotationVectors()

```
double [,] ParseVTKFormat.VTKPointDataReader.readRotationVectors (
             bool hasCorrectIDMapping,
             int [] correctedNodeID )
```

Here is the call graph for this function:


ParseVTKFormat.VTKPointData Reader.readRotationVectors
ParseVTKFormat.VTKgetName SpecificVTKDataArray.getNameSpecific DataArrayPointData

4.26.2.2 readTranslation()

```
double [,] ParseVTKFormat.VTKPointDataReader.readTranslation (
             bool hasCorrectIDMapping,
             int [] correctedNodeID )
```

4.26.3 Property Documentation

4.26.3.1 ExtremeDisplacement

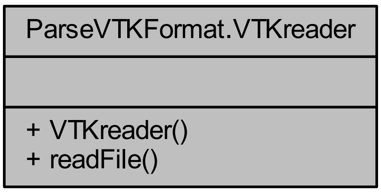
```
double [] ParseVTKFormat.VTKPointDataReader.ExtremeDisplacement  [get]
```

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

- VTKPointDataReader.cs

4.27 ParseVTKFormat.VTKreader Class Reference

Collaboration diagram for ParseVTKFormat.VTKreader:



Public Member Functions

- VTKreader (string filePath)
- vtkUnstructuredGrid readFile ()

4.27.1 Constructor & Destructor Documentation

4.27.1.1 VTKreader()

```
ParseVTKFormat.VTKreader.VTKreader (
             string filePath )
```

4.27.2 Member Function Documentation

4.27.2.1 readFile()

```
vtkUnstructuredGrid ParseVTKFormat.VTKreader.readFile ( )
```

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

- VTKreader.cs

Index