

CHALMERS



Developing a Graphical Post Processor

MARCUS GOBERT

Master's Thesis

Information Engineering Programme

CHALMERS UNIVERSITY OF TECHNOLOGY
Department of Computer Science and Engineering
Division of Computer Engineering
Göteborg 2007

All rights reserved. This publication is protected by law in accordance with "Lagen om Upphovsrätt, 1960:729". No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior permission of the authors.

© Marcus Gobert, Göteborg 2007.

Abstract

This thesis focuses on the development of a graphical post processor – a plotting tool which is used to display results from a solver used in gas exchange simulations. The tool is composed of two general parts: a step-by-step wizard that is used for the creation of the desired plot and a plot view which displays the selected data.

To ensure that the tool performs as expected in terms of functionality, performance, and usability, the development process has followed a non standard iteratively based process. This includes a pre-study followed by design, implementation, and testing phases.

The thesis will also discuss in-depth aspects of the design, such as the creation of software specific diagrams, design patterns used and design changes made. A few examples will be presented in order to demonstrate the functionality of the tool, and finally some conclusions are made from the project.

Sammanfattning (Swedish abstract)

Denna rapport beskriver utvecklingen av en grafisk postprocessor – ett plottningsverktyg som används för att visa resultat från en lösare för gasväxlingssimuleringar. Verktöget består av två generella delar: en steg-för-steg-wizard vilken används för att skapa plottar, och en plotvy som grafiskt visar den valda datan.

För att verktyget ska fungera som väntat när det gäller funktionalitet, prestanda och användarvänlighet har utvecklingen följt en icke-standardiserad iterativt baserad process. Detta har inkluderat en förstudie som följts av design-, implementation- och testfaser.

I rapporten diskuteras dessutom djupare aspekter såsom skapandet av mjukvaruspecifika diagram, designmönster och designändringar. Några exempel presenteras för att demonstrera verktygets funktionalitet och slutligen dras ett antal slutsatser från projektet.

Preface

This thesis finalizes my studies at Chalmers University of Technology for a master's degree in Information Technology specializing in Software Development and Management. This project was produced and conducted at Volvo Technology Corporation at Chalmers Teknikpark in Göteborg from October 2006 to April 2007.

Acknowledgement

I would like to thank my supervisor Urban Flock for providing me with this project as well as his invaluable support and suggestions made during the development.

Also, thanks to Sebastian Krausche, Christoffer Krausche, and Tobias Carlsson for providing me with sufficient ICES models used in the testing phases.

Table of Contents

Abstract.....	i
Sammanfattning (Swedish abstract).....	i
Preface.....	ii
Acknowledgement.....	ii
1 Introduction.....	1
1.1 ICES.....	1
1.2 ICES GUI.....	1
2 Background.....	2
3 Goal.....	3
4 Analysis.....	3
4.1 Curves.....	3
4.2 Maps.....	3
5 Method.....	4
5.1 Requirements Specification.....	4
5.2 Software.....	5
5.2.1 Visual Studio 2005 Standard Edition.....	5
5.2.2 Qt.....	5
5.3 Time plan.....	5
6 Results and Design.....	5
6.1 Class Diagram.....	6
6.1.1 GUI Module.....	8
6.1.2 Storage Module.....	10
6.1.3 Plot Module.....	12
6.2 Sequence Diagrams.....	14
6.2.1 Create Curve.....	14
6.2.2 Create Map.....	15
6.2.3 Update Plot.....	16
6.3 GUI Forms.....	17
6.3.1 Data Selector.....	17
6.3.2 Item Editor.....	18
6.3.3 General Editor.....	19
6.4 Design Patterns.....	19
6.5 Design Changes.....	20
6.6 Testing.....	21
6.7 Plot Examples.....	21
6.8 Future Work.....	26
7 Conclusions.....	26
8 Dictionary.....	28
Appendix A.....	30
Appendix B.....	31
Appendix C.....	32

1 Introduction

Modern gas exchange simulation generally requires a large amount of input and output data. In a simple running case, the user is often interested in a vast amount of fluctuating parameters. A slight change in one parameter can have major impacts on hundreds of others in many different parts of a simulation model. Therefore, in order to help users to manage the simulation process, there is a need to graphically display the changes of the output as well as comparing the data with other results. This project will focus on developing a plotting tool that can be used to visualize the output of any number of variables. In addition, the tool also needs to visualize special data structures used in the simulation process, called maps. The gas exchange solver is called ICES. The tool will be created as an extension of ICES GUI – a graphical interface for the solver.

1.1 ICES

ICES (Internal Combustion Engine Simulation) is a simulation program basically for 4-stroke Otto and Diesel engines and is written in FORTRAN at Volvo. A model based structure composed of components (engine objects), stations (measuring points), and connections between these are read into the program, typically from text files or from a console environment. Afterwards, the model is approximated in the solver using the options specified by the user. Finally, the output data can be fetched from output files or by using console commands.

Figure 1 shows a loaded model in the ICES solver.

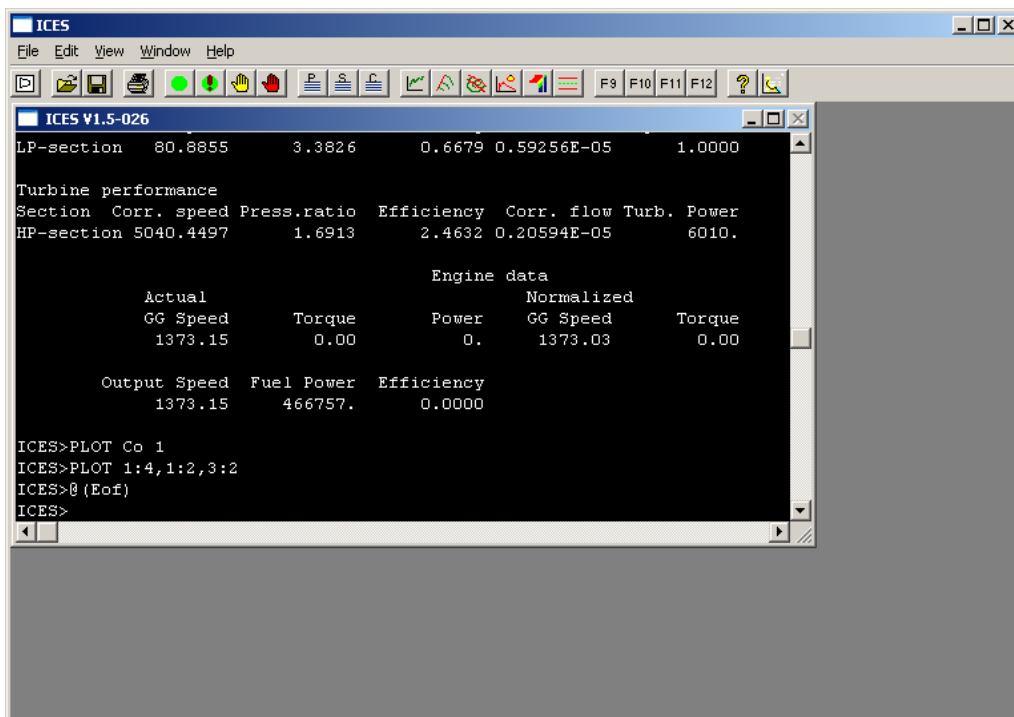


Figure 1: An image of the ICES solver.

1.2 ICES GUI

ICES GUI is an extension of ICES written in C++ at Volvo Technology which allows the user to graphically generate input models. Figure 2 displays ICES GUI with an open engine model.

Models are composed of model objects which in turn are composed of components, stations, or both. The model objects are connectable by using a mouse click environment. This program has been created by the author, starting in 2003.

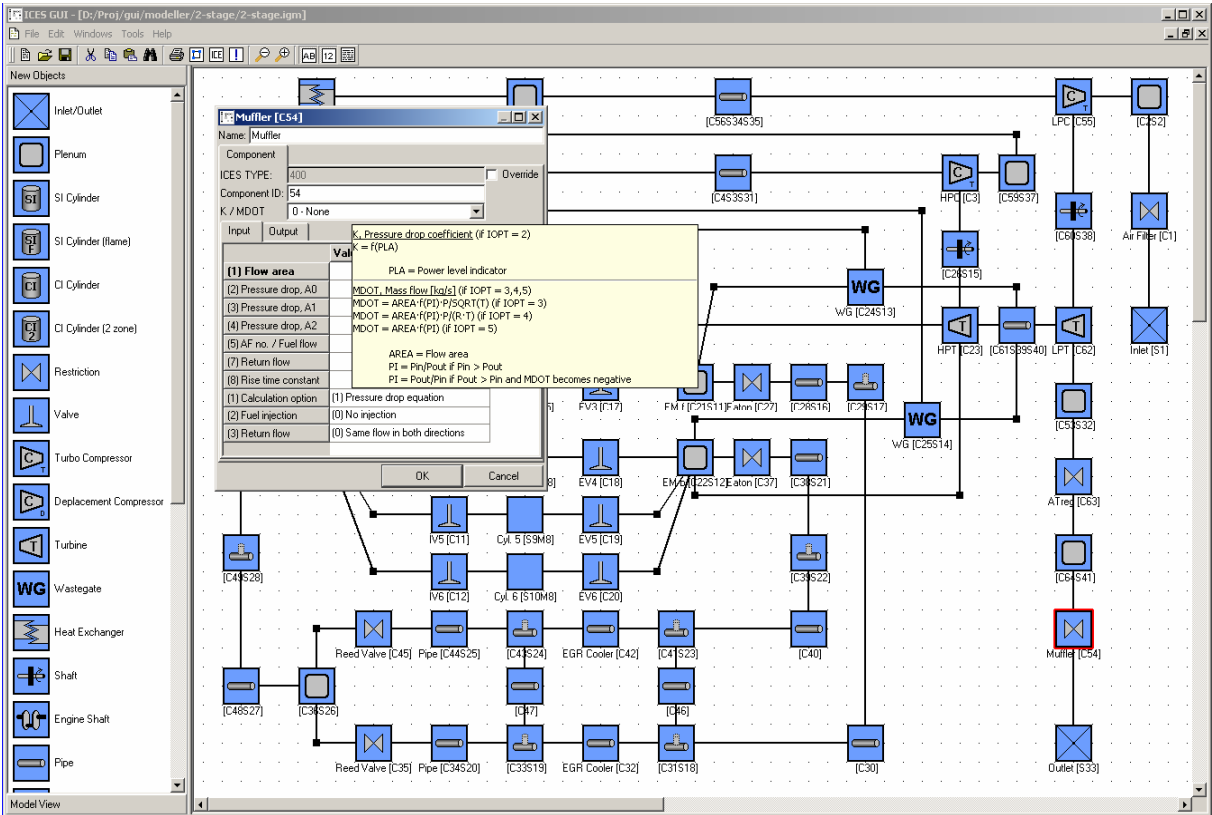


Figure 2: An image of ICES GUI.

2 Background

Editing text files in a text editor when creating or changing input models is generally a slow process. Even though ICES has a logical input structure, the input data tend to get large, thus, allowing for numerous logical errors as well as making it tedious to find the data that needs to be modified.

ICES GUI was created in order to make things easier for the user by speeding up the creation and manipulation of input models as well as reducing errors. Prior to this project however, ICES GUI only supported pre processing functionality (generation of input models). Some basic post processing (working with the data returned by ICES) was also supported in the sense that scalars could be examined. However, since the output returned by ICES is in most cases very large and most often varies depending on the time, there is a need to view the data in a more graphical manner. For example, temperatures in various engine parts are rarely constant during a cycle. Graphs fulfil the purpose of viewing varying data. However, in gas exchange simulation, it's often desired to examine how data varies depending on each other. Therefore, there must be an easy way of viewing and comparing different output. This work will focus on this problem.

3 Goal

The goal is to extend ICES GUI to support a graphical interface that can be used to view and compare varying output – a post processor. The tool must also include output features such as printing and exporting to image files. The interface must be easy to use and require as little work as possible to achieve the desired view.

4 Analysis

There are two major concepts that concern this project. The code needs to support both these in order to successfully display the needed graphical output. These are curves and maps and are explained in detail below.

4.1 Curves

In this project, curves are regarded as a sequence of 2-dimensional data points. These are connected in their order using lines. Curves are needed when the post processor needs to display the varying output of a value, such as cylinder pressure, during a cycle. A cycle is a sequence of time steps in each of which the ICES solver calculates an arbitrary number of variables selected by the user. These variables can be regarded as measuring points. The cycle is most often a complete 4-stroke cycle, although, there might be cases that does not include any cylinders.

As mentioned earlier, users often need to compare varying data. This can be done in two general ways. The first method is to compare both variables to the crank angle by using the respective variable as y values and the crank angle as x. This would require two curves. The second method is to compare them directly to each other by using the first variable as x values and the second one as y values. This would only require one curve. Note that any data could be used on both x and y, e.g. the crank angle can also be y data.

In similar to other plotting tools the curves should be distinguishable by using colour coding. Also, other visible properties should be available such as point markers, dashed lines or hidden lines. Finally, the curves should be visible in a legend, when desired by the user, in order to improve distinction further.

4.2 Maps

A map is a tabulated, numerical representation of a possibly multi dimensional structure. For example a map can be a set of measurements on a device for describing its characteristics. In ICES GUI maps for compressors and turbines are considered since these are often of great importance in gas exchange simulations.

Compressor and turbine maps in ICES consist of a number of curves that can be divided into two groups. The first group are curves that compose of 2-dimensional data. These make up a number of sections and the boundaries of the map. The second group consists of efficiency curves. These are level curves within the boundaries that are calculated in a 3-dimensional structure where the level is the z axis data. Finally, there might be an optional operating curve. This curve displays the operating points of the current cycle. In addition, this curve also needs to display where the starting and ending points are located.

The plot view needs to support visualization of compressor as well as turbine maps. Compressor maps can represent both turbo compressors and displacement compressors. Turbine maps can represent turbines of single entry, dual entry, or VGT types. Although every one of the above types looks different, they are all generated using two or three convenient 2-dimensional or 3-dimensional table structures from ICES. Since the table sizes might differ and since the resolution (the number of rows and columns) might be low, the tables are resized, normally to 50 rows and 50 columns. This is done by using bilinear interpolation and extrapolation.

In order to make the curves within a map distinguishable, they will be both colouring coded and have their own labels within the plot area. In addition, the legend will display all the curves and put them into their respective categories.

5 Method

The project has not strictly followed any general software development process such as RUP or XP, however an iterative work flow including pre-study, design, implementation, and testing has been used. The main argument for this is that it was a one-man project. Although, the project is quite large, there are still several benefits by constructing a proper design and using proper testing methods.

The pre-study phase included creation of the software requirements specification, deciding which tools and libraries to use and creating a time plan. All these aspects are described in detail below in this chapter.

The design phase included creation of UML class and sequence diagrams and GUI models. Diagrams and models like these facilitate an over all picture of how the different parts work and how they interact. This is a critical part of the project since a good design often leads to less code reconstruction and bugs. Therefore, a lot of time was spent on this phase. The implementation phase however, was the largest and included the actual coding and creation of the manual. Some reconstruction of the code had to be done due to various reasons.

Details of the design, implementation, and testing phases are presented in the Results and Design chapter.

5.1 Requirements Specification

All functional requirements in the requirements specification were mainly developed together with the project supervisor. Some ICES users also gave their views of what aspects program should fulfil. The functional requirements were not described using use cases, but instead by using a punctuated list. The reason for this is that the requirements were general and therefore quite trivial. The non-functional requirements are usability and performance related. Usability issues were dealt with in the design phase and performance issues were mainly handled directly when coding.

The Requirements Specification is included in Appendix A.

5.2 Software

The main software used during the development of the project, the IDE and the programming library, are described below. In addition, other software such as Microsoft Excel 2005, Microsoft Word 2005, Paint Shop Pro 5, FinePrint (a printer tool), and Notepad++ (a tabbed text editor) were utilized. The revision control system utilized during the whole project was subversion using the TortoiseSVN client for Windows.

5.2.1 Visual Studio 2005 Standard Edition

The IDE used for the project was Microsoft Visual Studio 2005 Standard Edition. This choice was logical since this is a commonly used coding environment at Volvo Technology, and used for the ICES GUI code.

5.2.2 Qt

Qt is a cross-platform programming library developed by Trolltech that mainly contains classes for GUI-programming. However, it also covers areas such as networking, graphics, serialization, and IO-streams. Additionally, it contains a verity of core classes for strings and data containers. This project, however, will only focus on the GUI, graphics, and some of the core classes. The Qt programming library is also used by the ICES GUI code.

5.3 Time plan

A general time plan was made up by estimating the time cost of the requirements in the requirements specification. This was done directly after the pre-study and extended early in the first design phase when more data had been collected of the actual size of the project. In general, most time constraints have been met, though, there are some overdue in the first iteration. The major reason for this was that the current version of Qt was a release candidate. This version hade some minor bugs which had to be circumvented in order for the tool to function as expected. The creation of maps in the second iteration is also overdue. This process was harder than expected. These over dues delayed the work and therefore, the creation of a standalone application (scheduled in iteration 3) was postponed for future development.

The time plan is located in Appendix B.

6 Results and Design

This section contains the final design from all iterations. Firstly, the class diagrams created for the tool are described in order to get an overview of the code. To provide an in-depth description, a few sequence diagrams are explained for the most common operations within the program. To provide a visual display of the graphical controls within the tool, the wizard interface is described as well. Finally, general design concepts are described, such as design patterns and testing.

6.1 Class Diagram

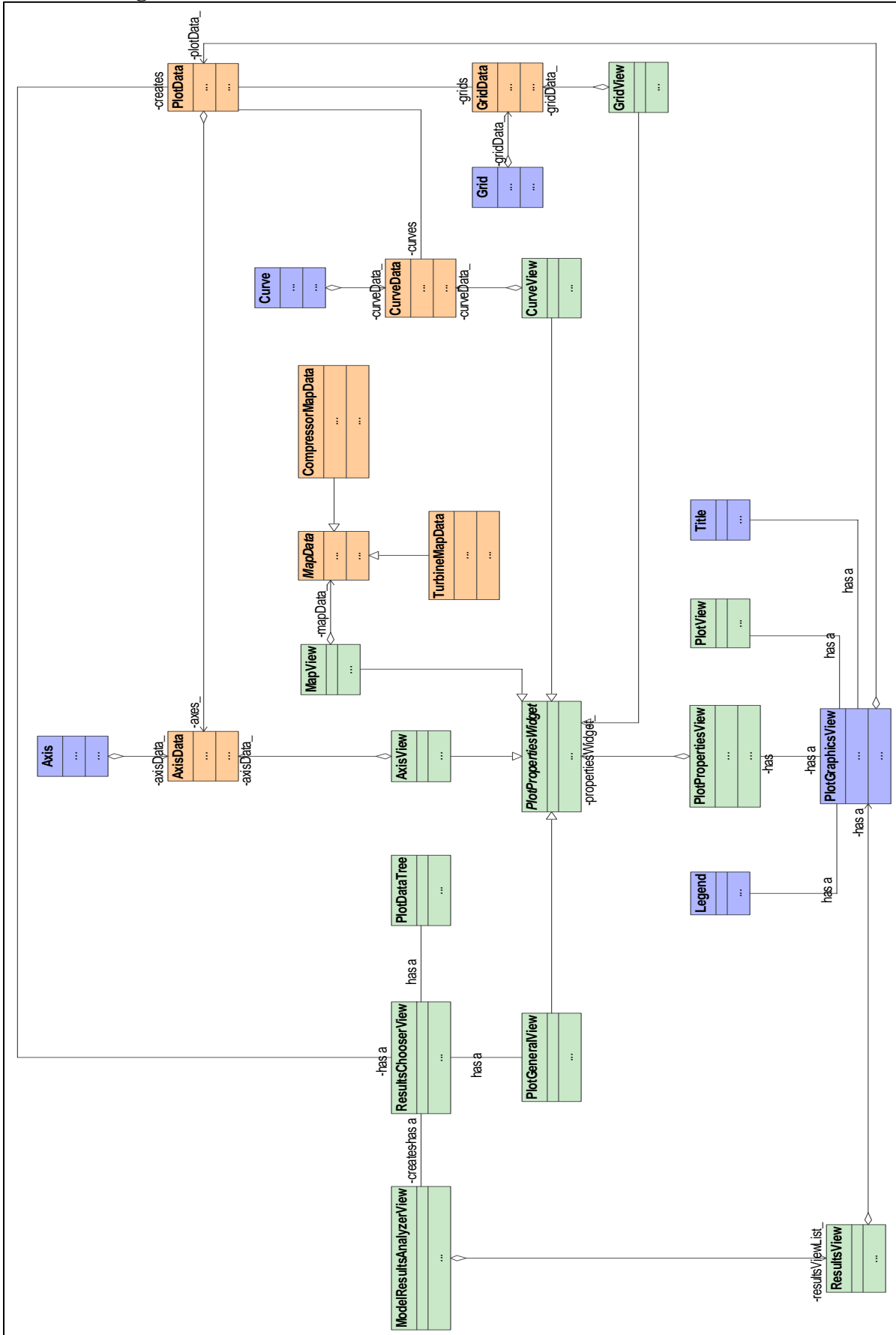


Figure 3: The complete class diagram showing all 3 modules in the post processor.

The class diagram in Figure 3 shows the interaction of all classes created in the project, however, due to space constraints, all functions, instance variables and Qt classes used in the project have been removed.

The diagram has been divided into 3 modules coloured in red, green and blue. The red classes belong to the storage module. These function as data storage as well as some data generation. The green classes a part of the GUI module and inherit one or more Qt GUI classes. These handle all user interaction in terms of viewing as well as manipulating the classes in the storage module. Finally, the blue classes indicate the Plot module which handles the drawing of the plot area. They inherit the Qt graphics view framework.

The modules are explained in detail below.

The GUI Module classes are explained in detail below. The relationships between them as well as their internal structures are shown in Figure 4. Every class inherits a Qt GUI component (not shown in the class diagram).

ResultsChooserView

This is the main widget class for choosing which data to plot and how to plot it. It is a wizard structure containing 3 pages. The first page is the actual data chooser consisting of 2 graphical trees. The first tree contains the data that can be added to the plot and the second tree contains the data that has been chosen for the current plot. The second page is an index of the plot items (curves, axes, grids and maps) that has been generated by the system using the chosen plot data. These items can be modified accordingly. The last page handles the general plot options such as the title and the legend. These options are optional.

The 3 pages can be navigated backwards and forwards by using the “Back” and “Next” buttons. In addition, the plot operation can be cancelled by pressing the “Cancel” button. The plot is generated when the “Finish” button is pressed.

PlotDataTree

This is the graphical tree that contains the data that can be plotted. It inherits QTreeWidgetItem (not shown by diagram) which enables various tree functions from Qt. This class also receives a pointer to the output structure from ICES, ModelResults. The tree scans the data and translates it into a user friendly tree structure using QTreeWidgetItem.

PlotView

PlotView represents the graphical window that contains the plot graphics class, PlotGraphicsView. It is created by ResultsChooserView when the user presses the “Finish” button. This view can be used in 2 modes; selection mode and measure mode. In selection mode the view will allow the user to select specific items and change their corresponding properties. Measure mode enables mouse tracking and allows the user to click within the plot area and get specific readings at the mouse position. This information is shown on the axes.

PlotPropertiesView

This is the graphical window used by PlotView to create an appropriate options window when the user wishes to change an item in the plot area by receiving a PlotPropertiesWidget.

PlotPropertiesWidget

PlotPropertiesWidget is an abstract class that enables PlotPropertiesView to call the save and update functions on the current properties view. This class is extended by CurveView, AxisView, GridView, MapView and PlotGeneralView.

CurveView, AxisView, GridView, MapView, PlotGeneralView

These classes create appropriate widgets to change the specified plot. In addition, they implement PlotPropertiesWidget with update and save functions. They are created by both ResultsChooserView which enables modification in the wizard and PlotView which enables modification when the plot has been created.

ResultsView, ModelResultsAnalyzerView

These classes enable the use of several views when plotting or listing scalars in the same window. They have not yet been implemented. Read section “Future Work” for more information on these.

6.1.2 Storage Module

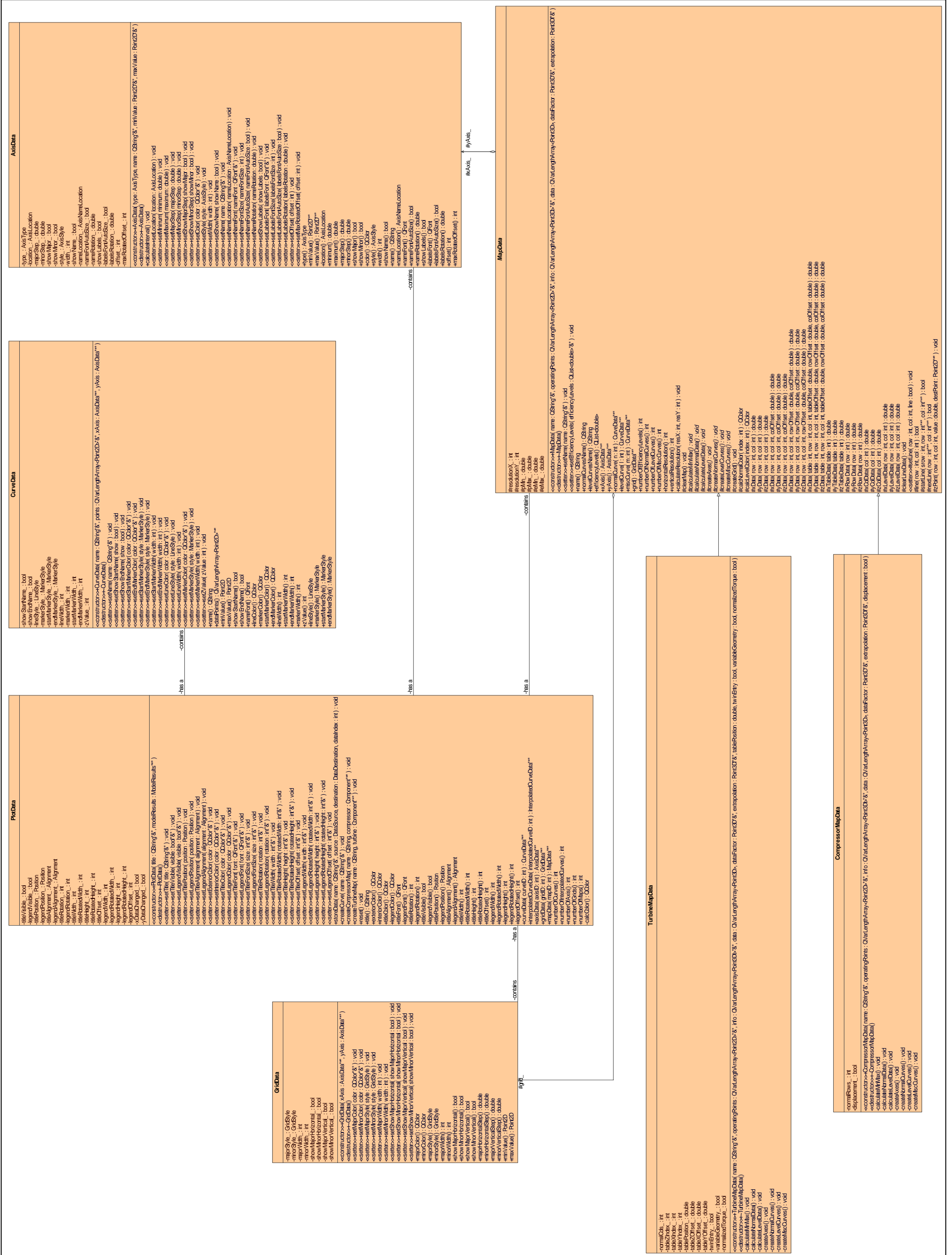


Figure 5: A class diagram of the Storage Module showing their structure and relationships.

As can be seen in Figure 5, the classes in the Storage Module contain many functions and instance variables. The reason for this is due to the extensive amount of options that can be made to the plot view. Therefore, most of the functions are set and get operations on the instance variables used for the properties of the plot view.

PlotData

PlotData is the main data container for the plot, however, it also acts as a factory for the creation of all other sub containers; CurveData, AxisData, GridData, CompressorMapData and TurbineMapData. The data is fetched using a pointer to the output structure from ICES, ModelResults. In addition, it contains optional general plot data such as a title and a legend.

CurveData, AxisData, GridData

These are the data containers for the plot items. They contain visual options such as colour, font and location. The CurveData class also contains the data points read from the ICES output structure. Each item is given a reference to a corresponding data container and then the plot items are read from the containers and drawn accordingly.

MapData

This is an abstract data container used by the maps. Incoming data is translated to a more convenient 2-dimensional table structure. In addition, several other values, such as multiplication factors and size constraints are stored in appropriate lists. Finally, it provides protected functions for the inheritance classes for convenient retrieval of the data. This class has been generically programmed to allow for easy supplementation of new map types.

CompressorMapData, TurbineMapData

These are large classes who facilitate the creation of the actual map data and items using the specified options. The final map data is calculated from the data in MapData using bilinear interpolation and extrapolation. The level curves are generated by using the same point search algorithm used by ICES.

Every class in the Plot module inherit a Qt Graphics component. The PlotGraphicsView inherits the QGraphicsView and all other classes inherit QGraphicsItem. Figure 6 displays the complete class diagram of the Plot Module.

PlotGraphicsView

PlotGraphicsView is the main view of the plot however; it also handles the creation of the plot items (Title, Legend, Curve, Axis and Grid) by supplying each item with a corresponding container. Each time the view needs to be updated (by resizing it for instance) the updateItems function is called. This function resizes the curve area (the area covered by the axes) by calculating the margins needed for all axes, the title and the legend. Finally, the items themselves are updated using the margins calculated. The actual updating the items are not done directly, but instead, Qt puts them in a queue when the update function of the items gets called. Items in the queue are then updated when possible. Since updateItems is called very often when resizing, much effort has been spend in order to make it as fast as possible.

Title, Legend

These plot items handles the drawing of the title and the legend using the margins specified by PlotGraphicsView. The legend also calculates inner margins by looking at the number of curves in the plot. Both classes are optional since it's not always necessary to have them.

Curve, Axis, Grid

These classes handle the drawing of the visible plot items using the specified data containers (CurveData, AxisData and GridData). In addition, they are supplied with margins from PlotGraphicsView in order to calculate the necessary point translations and locations. They all inherit QGraphicsItem to allow for drawing specific functions called by Qt. Before painting rotation and translation operations are made on a 2-dimensional transformation matrix.

6.2 Sequence Diagrams

This section contains sequence diagrams of 3 critical parts of the code, the creation of a curve and a map in the PlotData object and the update procedure of PlotGraphicsView object.

They are explained in detail below.

6.2.1 Create Curve

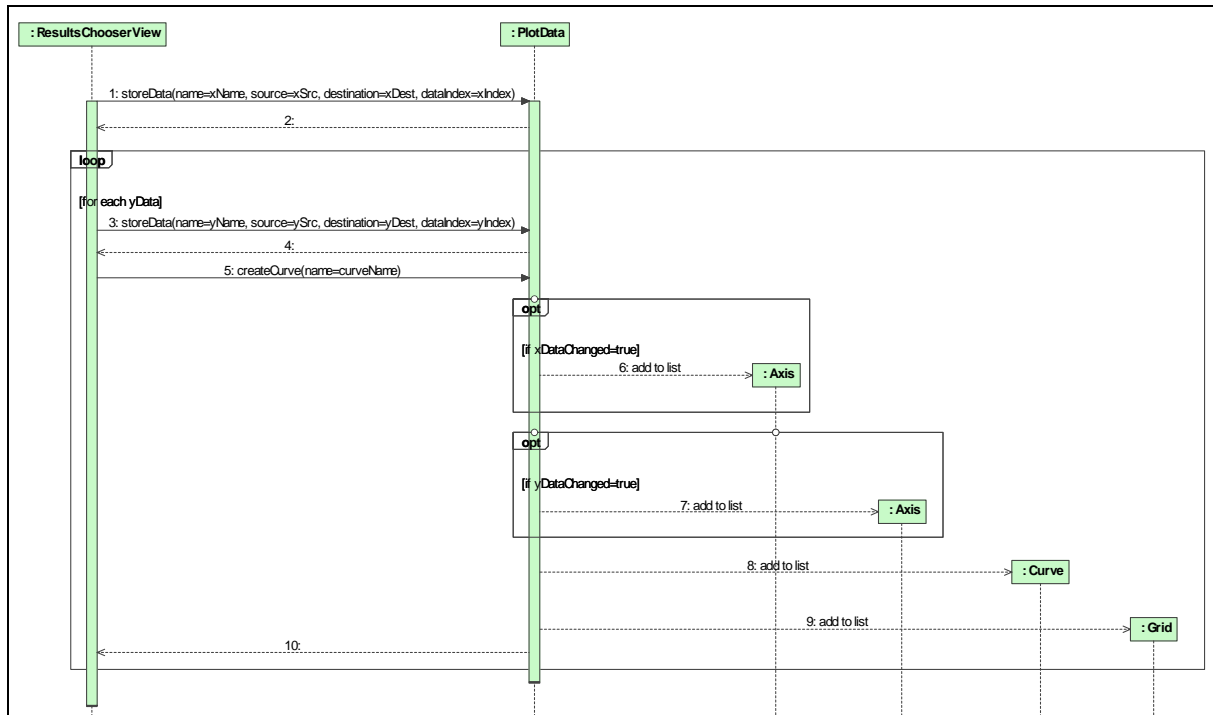


Figure 7: A sequence diagram of the create curve operation.

As can be seen in Figure 7, the creation of a curve in the PlotData container will allocate 3 or 4 plot items, a curve, a grid and 1 or 2 axes depending on the number of `storeData` calls that have been made. The `storeData` call saves the data in a temporary buffer in the PlotData object. This data is then copied to the curve item when `createCurve` is called. Note that the sequence diagram first calls `storeData` with x and then loops through each y, however, it's also possible to do it reverse, thus, first by calling y and then looping through each x. This sequence can be clarified with an example: suppose the user wants to plot 2 curves, cylinder temperature and pressure as a function of the crank angle. Therefore, the user will create a top tree item (crank angle) and two leaf items (temperature and pressure) in the PlotResultsChooser. This will generate one `storeData` call for the crank angle and two `storeData` and `createCurve` calls in the loop, creating 2 curves, 2 grids and 3 axes.

6.2.2 Create Map



Figure 8: A sequence diagram of the map creation operation.

Creating a map is different than creating a curve since the data storage operations are made within the createCompressorMap or createTurbineMap calls. The reason for this is that maps have predefined x and y axes, therefore making it unnecessary for the user to make such a choice. The sequence diagram shows the creation of a compressor map. Note that the first 2 storeData calls stores the actual map data, whereas the optional storeData calls stores the operation curve. The storing of the data is done in the same way as curves, in a temporary buffer and then copied to the map. Figure 8 shows the map creation operation.

6.2.3 Update Plot

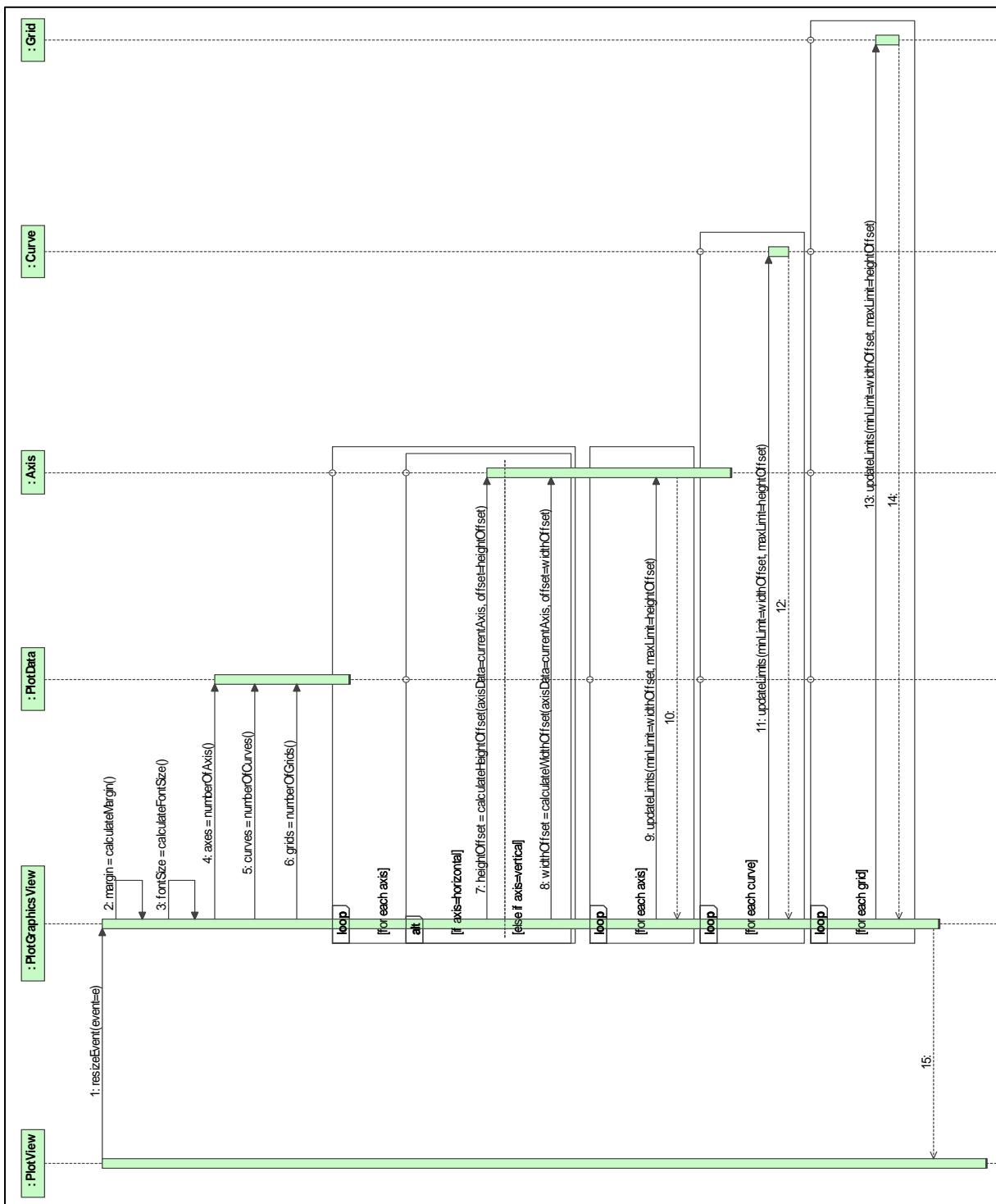


Figure 9: A sequence diagram showing the Update Plot operation.

The plot is updated when the `PlotGraphicsView` object receives a resize event. This is automatically made by the signal slot mechanism in Qt. Afterwards, the outer margins and the font size are calculated depending on the new size of the view. The view will then iterate over all axes in order to calculate their positions and sizes. Finally, all the plot items are updated using the offsets previously calculated. These operations can be seen in Figure 9. Note that the `updateLimits` call also puts the corresponding item on the update queue. These items are then redrawn automatically when Qt finds an opportunity to do so.

6.3 GUI Forms

This section contains the GUI forms for the plot wizard. The tool used to create them was QtDesigner which is supplied with Qt. It enables users to add various Qt widgets on forms using a drag and drop interface. This allows for easy creation of graphical interfaces. QtDesigner form-files can be compiled directly into the project using qmake (a qt compiler) however, this feature was not used.

The wizard consists of 3 parts; a data selector, an item editor and a general editor. The parts are accessed with the “back” and “next” buttons in the wizard however, in most cases; users will only use the data selector since much of the work is done automatically by the code.

All parts are explained in detail below.

6.3.1 Data Selector

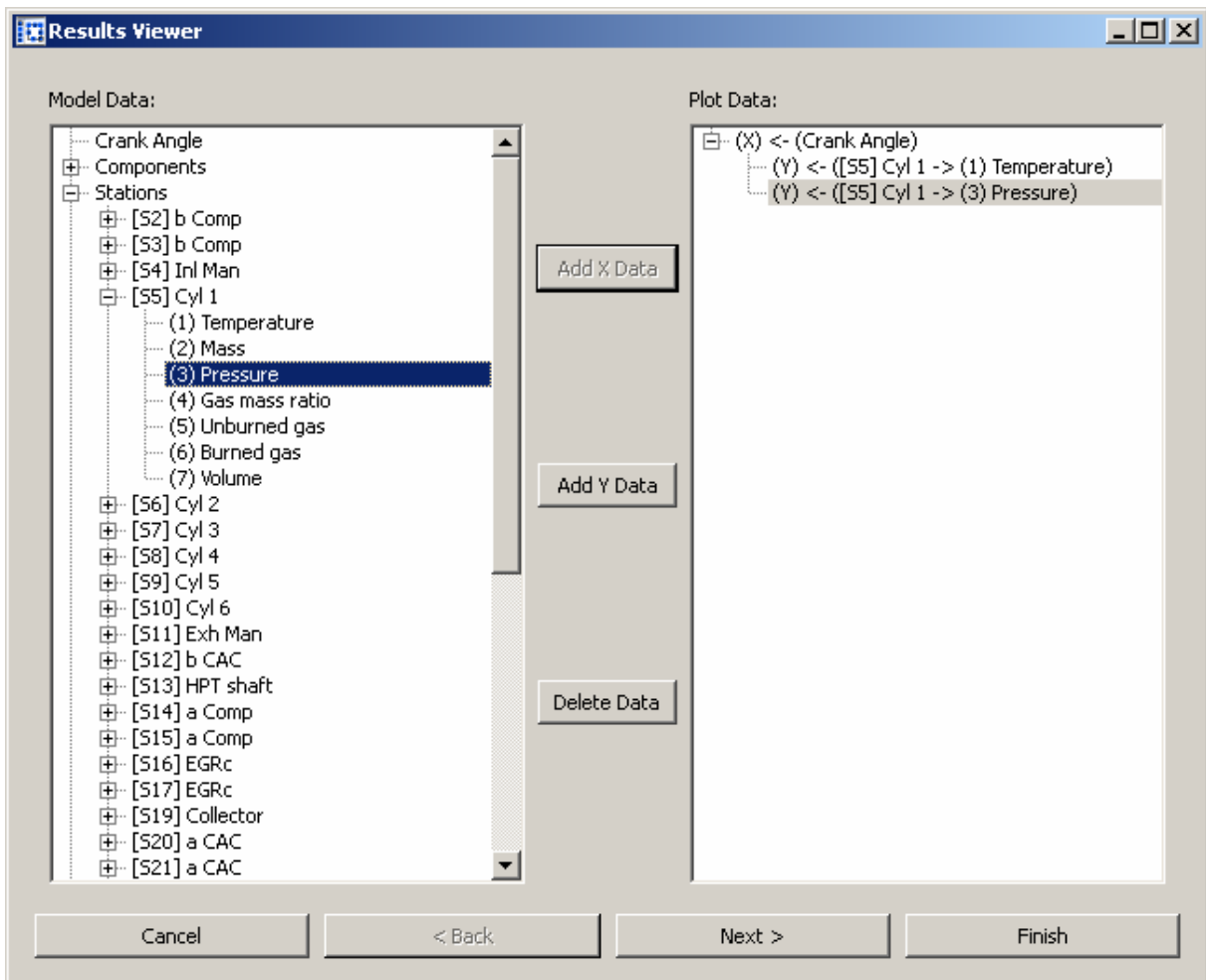


Figure 10: An image of the Data Selector page in the Results Viewer wizard.

The data selector is the first page in the wizard. This is shown in Figure 10. It enables the user to choose the data that has been outputted by ICES from the ModelResults object. The left tree contains the data that can be used for plotting and the right tree contains the data that has been chosen for the desired plot. By using the add data buttons the user can add the selected data in the left tree to the desired destination, x or y, in the right tree.

6.3.2 Item Editor

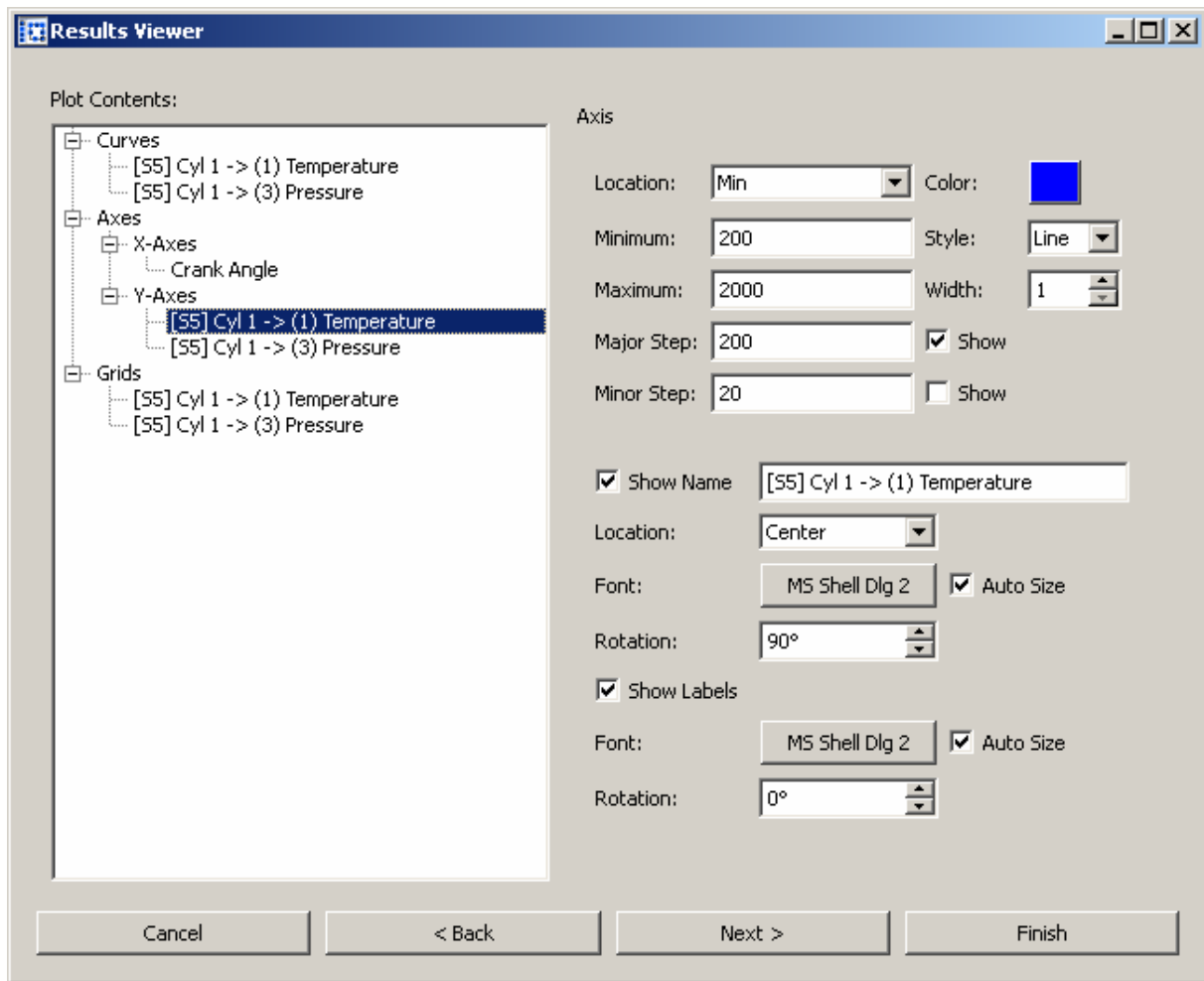


Figure 11: An image of the Item Editor page in the Results Viewer wizard.

The item editor, shown in Figure 11, lists the items in the tree to the left, which have been created from the selected data in the data selector. They are grouped by their respective types; curves, axes, grids and maps. Selecting an item in the tree will show a widget to the right where modification of the item can be made. These item modification widgets are also used when the user wants to modify a specific item in the plot view.

6.3.3 General Editor

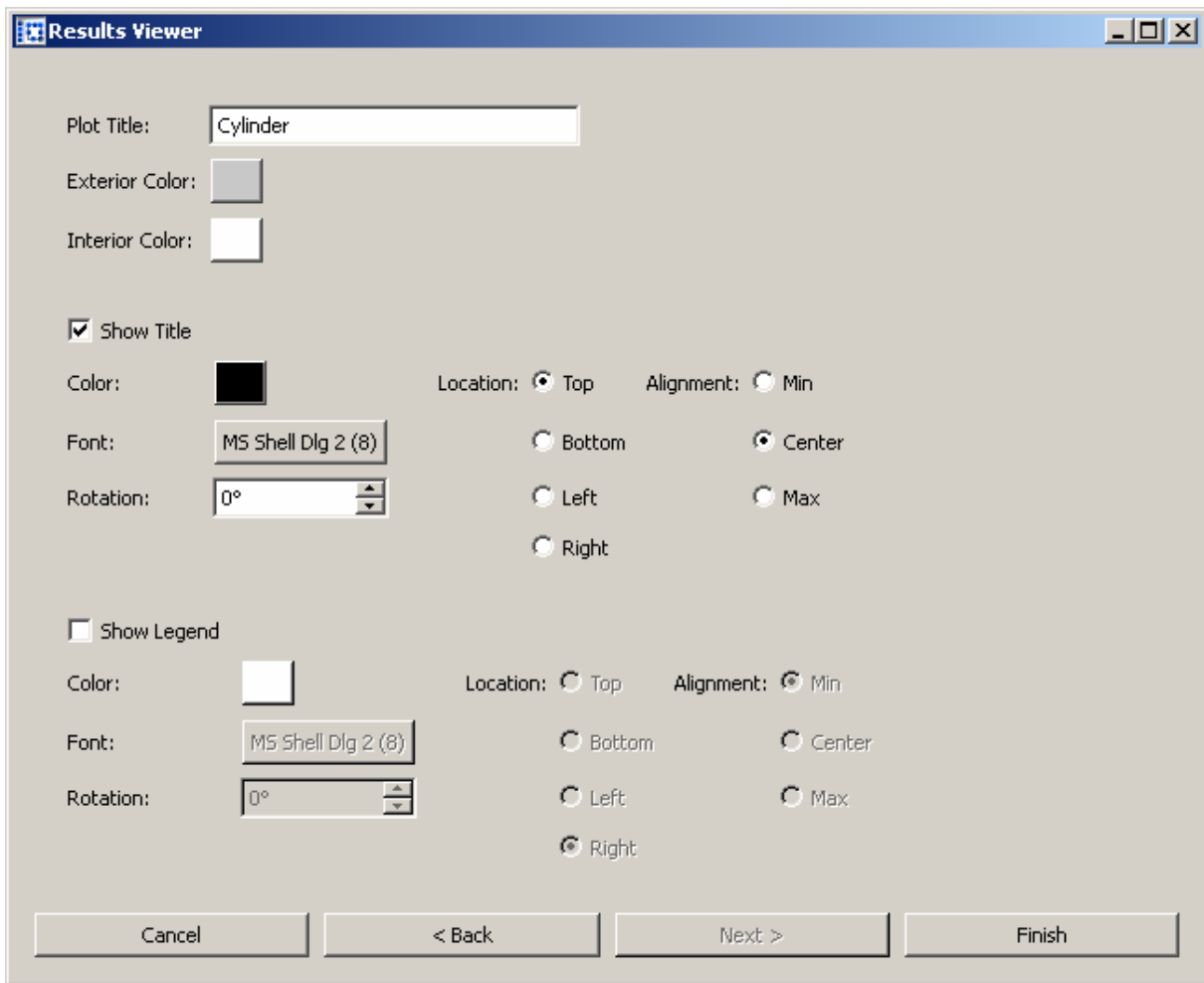


Figure 12: An image of the General Editor page in the Results Viewer wizard.

The general editor is the last page in the wizard. This widget allows the user to change non-item specific properties of the plot, such as title and legend. These options can be accessed in the plot view as well. Figure 12 displays this page.

6.4 Design Patterns

A future goal has been to create a stand alone plotting application similar to this project. This follows that there should be much gain - in terms of cost and time - of being able to reuse much of the code created with this project. Therefore the main design principle used for the duration of the project has been to design the code as generically as possible. In order to fulfil this goal, a few design patterns have been applied to the design.

The graphical classes within Qt have a Model/View architecture to manage the relationship between the data and the way it's presented to the user. This is a simpler case of the Model/View/Controller pattern in which the Controller part has been embedded into the View. Note that this does not abandon the separation concept since the data part is still separated from the view. The Model is the interface that views use to access its underlying data and the view obtains items from the Model and presents them to the user. Since the Controller now is embedded into the View, the code uses the Qt signal/slot system in order to communicate with the different parts e.g. the views with the models. Qt also introduces the

concept of delegates. These improve the way users can customize the way the data is edited, however, no such widgets were applicable for this project. In this design the storage module can be regarded as the Model and the GUI and Plot modules are the Views.

The graphical items in the plot view are parts of the whole drawing area. Therefore, these items have been structured as a simple composite pattern – a tree with a root node (the drawing area) and its sub items (curves, axes and grids). In order for the QGraphicsView to draw its items all sub items must inherit the abstract class QGraphicsItem. Three functions are implemented. The first is the paint function. This function gets called by the QGraphicsScene class when the item has been scheduled for an update. The second is the boundingRect function. This function returns a QRect structure (a rectangle) which tells the scene roughly where the item is located. By roughly, the meaning is that not all items are rectangular. A circle item would for example return a rectangle with its width and height equal to its diameter. The last function is the shape function which returns a QPainterPath structure. This structure is a list of shape objects, such as rectangles, circles or lines, that all cover the whole item. The reason for having both boundingRect and shape is that boundingRect is generally much faster and can be used for fast checking of collisions between QGraphicsItems. Meanwhile, the shape function is much more precise.

The custom made widgets that handle the modification of plot items (CurveView, AxisView, GridView, MapView and PlotGeneralView) have two things in common. Firstly, they contain a function that updates the widget using the storage object received and secondly, they all need to save the current state that has been chosen by the user. Therefore, all these widget classes implement a decorator pattern. The decorator is the abstract class PlotPropertiesWidget. This is used both by the wizard and the plot view.

6.5 Design Changes

Even though much effort has been put on the design of the code there were still some parts of that needed to be changed. When the project started the official version of Qt was still 4.1. This version did not contain the new graphics library that has been used for this project. However, a release candidate of version 4.2 was available. This version did contain the libraries needed. Although, using the old version 3 of Qt with the old graphics classes would have been possible but since ICES GUI had been developed by the later version, this was not an option. The release candidate version however contained some bugs and as time passed by and new official versions were released, minor changes were made to the library which in turn affected the code - mostly for the plot view. These changes did not however change the design drastically, only functions were affected.

The first version of the design only contained a Map class instead of two classes inheriting the Map class. However, in the coding phase it became clear that using inheritance was a necessary concept since a compressor map and a turbine map differs in many ways. Using this structure also makes it easier to add new map types and modifying the existing ones. Reading the data however, is done in the same way by both map types and therefore this code was not moved to the inheritance classes.

An issue that was discussed in the pre-study was the need of custom views of scalars (data that remains constant during a calculation). These could have been added by the user and displayed in a table structure. Therefore, the first design contained scalar views and table widgets. This idea was postponed due to the time constraint within the project.

6.6 Testing

A typical software test plan has not been created for this project, however, during development; the code has been undergoing extensive testing. In order to make sure the tool fulfils the requirements in the requirements specification, that it's stable and that it's usable on typical machines and environments, both unit, integration, functional and performance testing have been applied to the development phase. A typical workflow has been to implement a few lines followed by a small test and then repeat these 2 steps. Large functions have been tested more thoroughly with several different inputs.

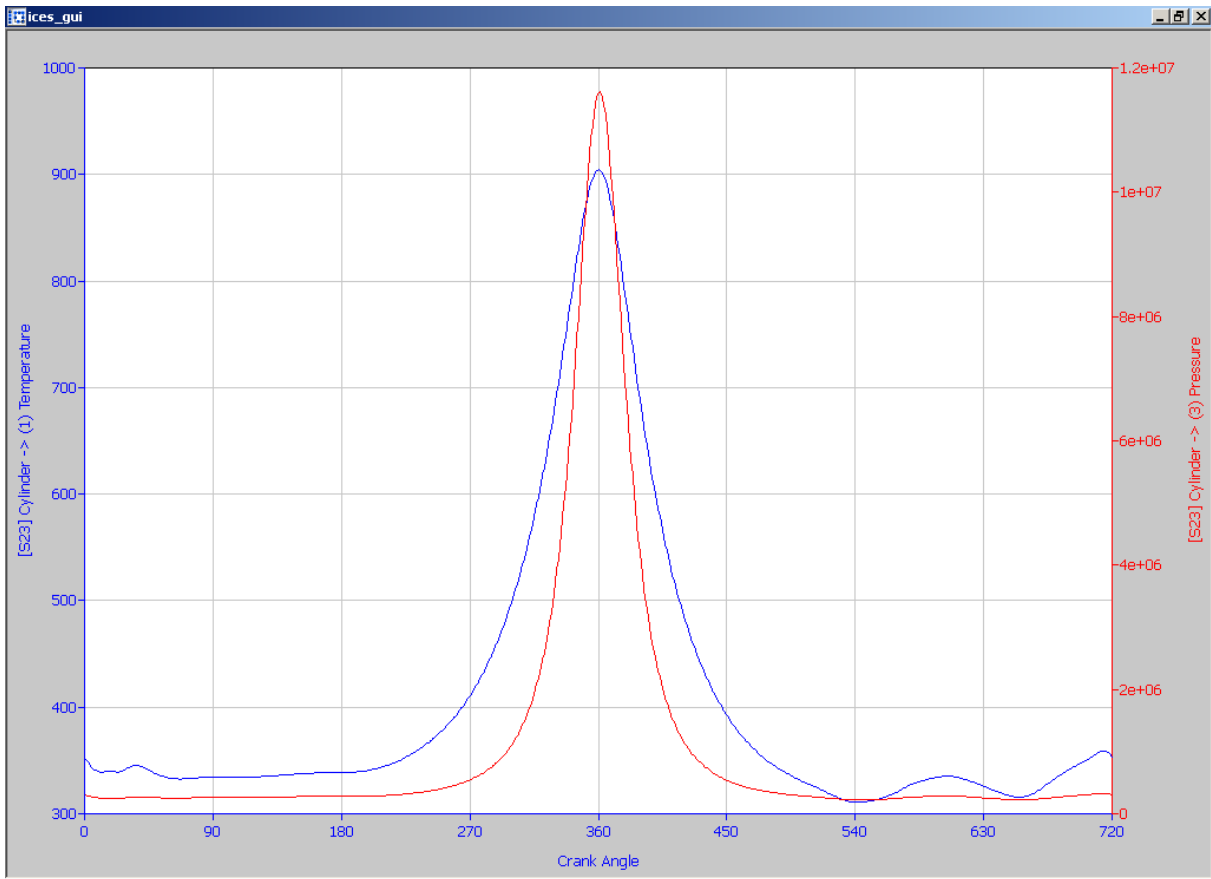
The white box testing has been accomplished by unit and integration testing. An early decision to manage the unit tests was to use an automated unit testing environment. There are many of these available for the C++ language and the one selected for the duration of this project was CppUnit which is open source and based on the popular JUnit for Java. The reason for this was that it also contains a special extension usable on Qt developed code, called QxTestRunner. This turned out to be a very good tool since it was fast, very flexible and easy to use. The integration tests were done manually and mostly included testing between the communication of the graphical classes and the storage module. This was also done directly while coding.

The functional testing part has included black box testing. This was only done from a user perspective and included two major parts. Firstly, many tests were made with the wizard. This included tests to insure that the correct plot items were generated with the desired plot and that the modifications made to the plot were stored accordingly. The second and largest part included testing of the actual plot view. A large amount of different plots were generated using many different models and parameters. To ensure their correctness they were compared with the data viewer in ICES.

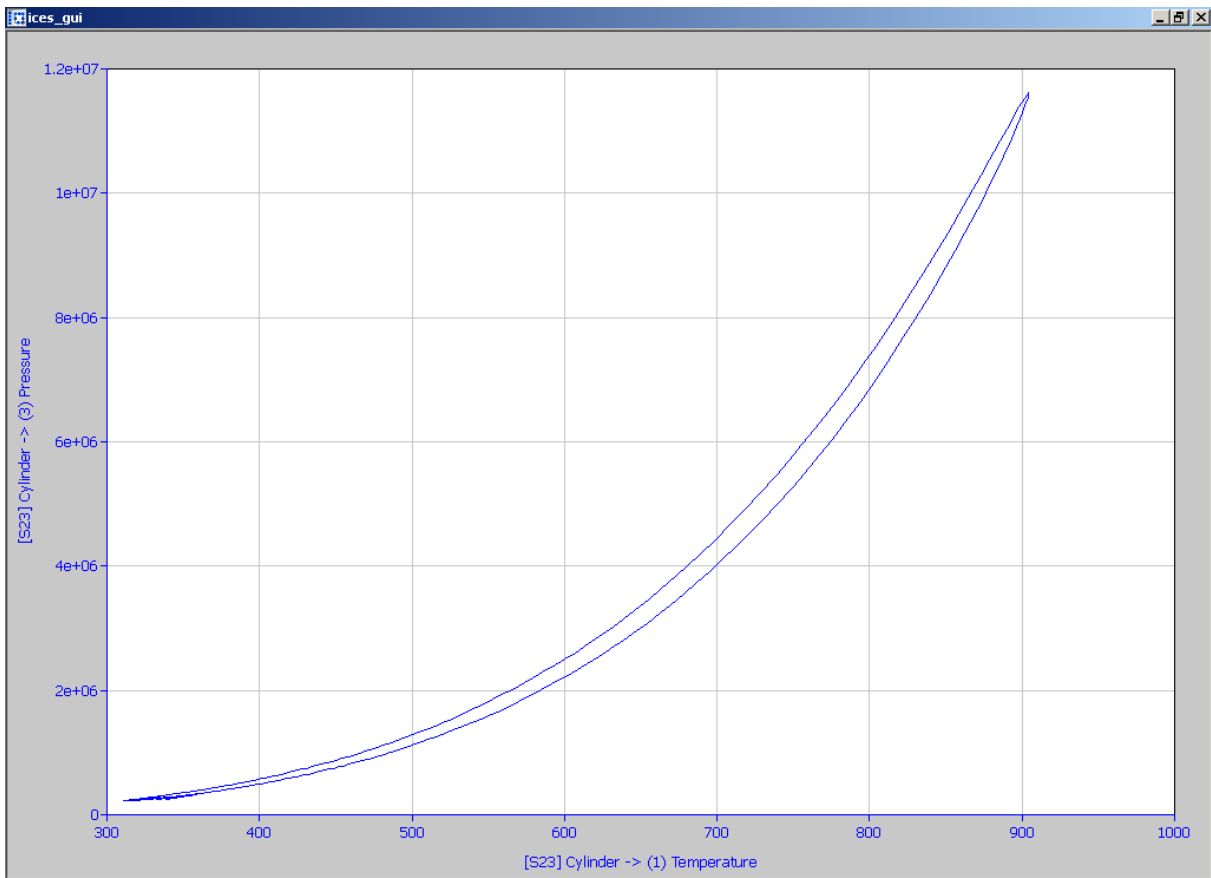
The final test made was performance testing. Since this plot view might include many items there is a need to ensure fast drawing of the items in order to ensure usability. A large impact on the performance of the plot view had to do with how the boundingRect and the shape functions of the plot items were implemented. In the early phase of the project, they were implemented in a manner which forced the view to update all the items even though only a small section of the view had been changed. Even though the view was indeed very fast due to Qt's fast drawing framework, there was a noticeable change of the performance after some optimizing of the above functions when drawing multiple items. On the systems used for development this change became noticeable with views with more than 15 curves in the same plot, a case most applied to maps. However, this system was very modern at the time of the writing and on slower systems the change might be more noticeable. The final performance tests were made to the actual generation of the plot. A plot is generated when the user has selected the data to be plotted in the wizard and press "next" or "finish". This is a critical performance dependent part since its dependent on much data throughput and generation. To avoid unwanted waiting times the affected code have been optimized, especially for the map generation since this is by far, the heaviest process. On the system used for development the generation of 5 maps using standard settings takes under 0.5 seconds, a reasonable waiting time considering only one map per plot is preferred.

6.7 Plot Examples

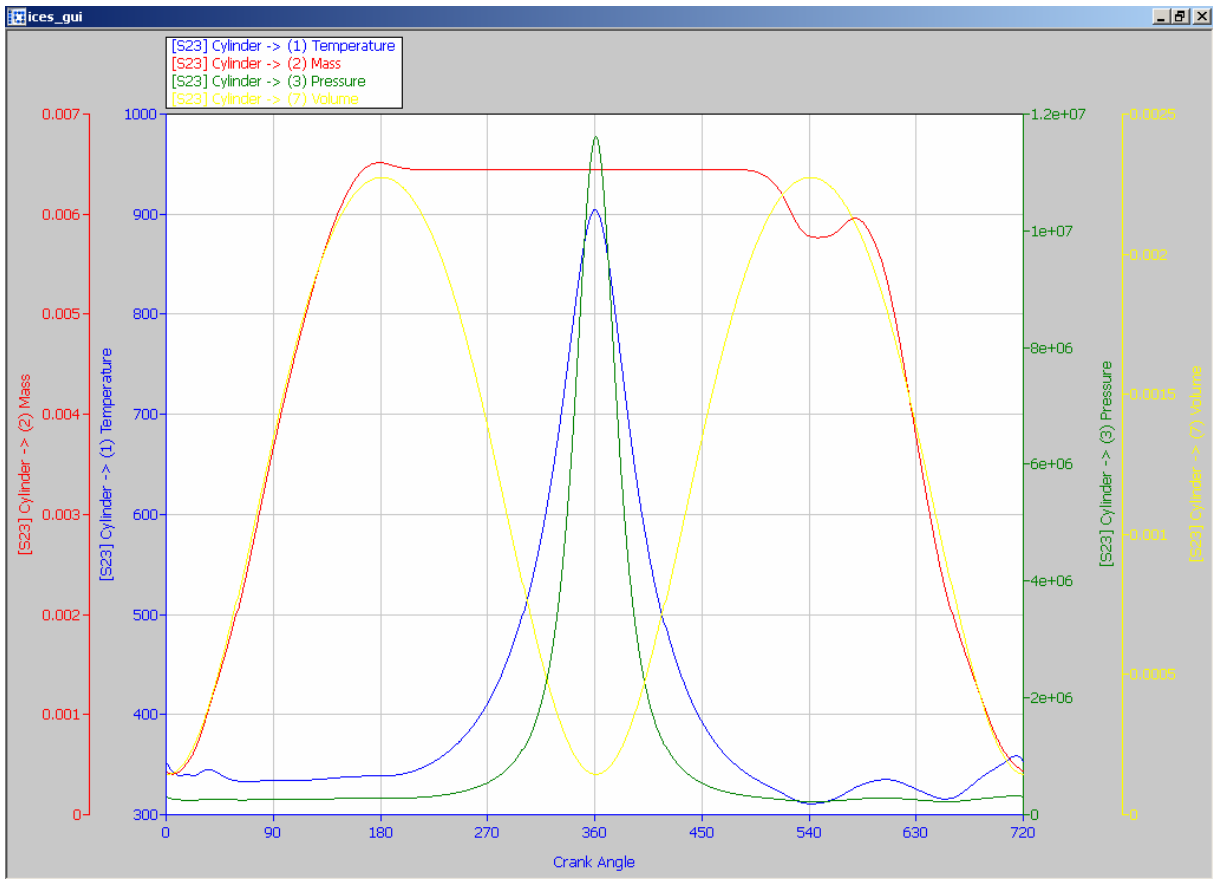
This section contains a few plots that have been generated with the post processor using different ICES models.



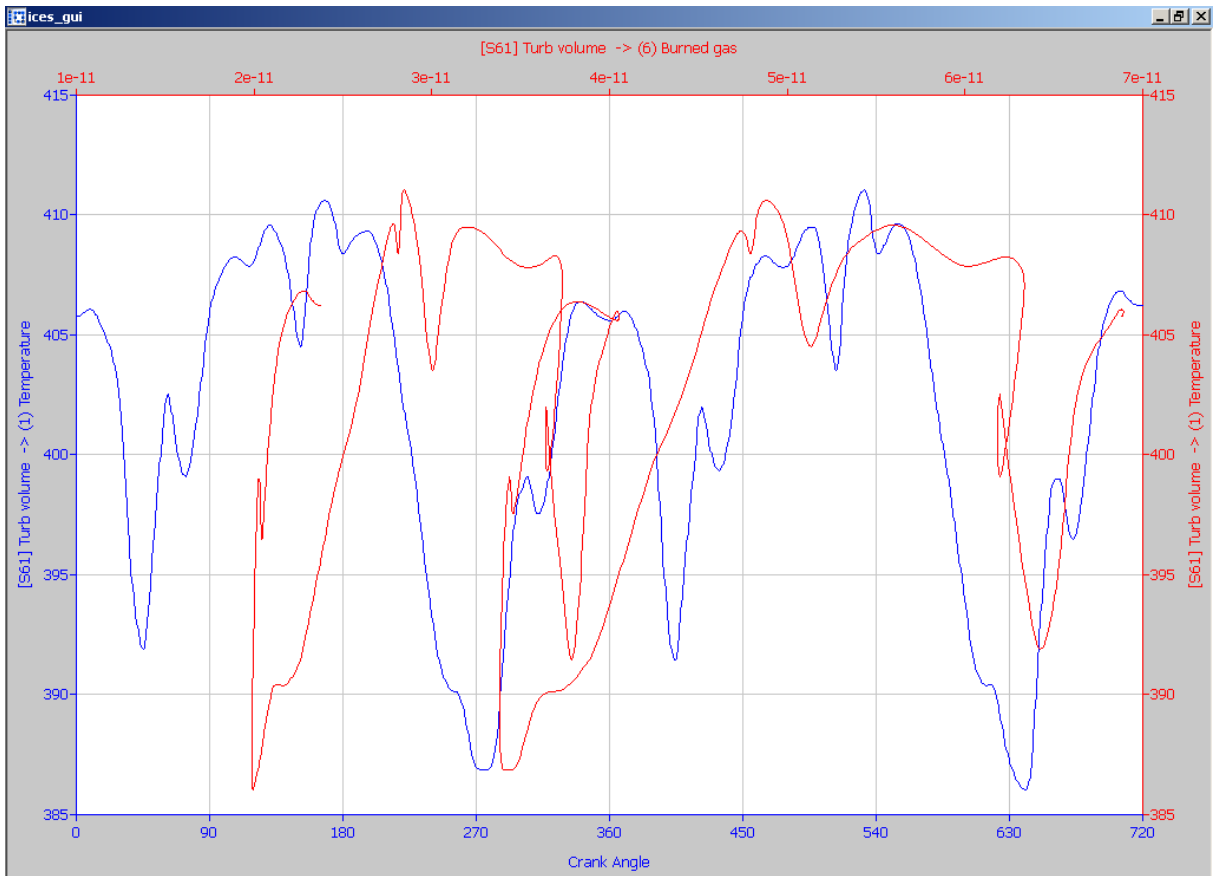
A plot showing the variations of the cylinder temperature and pressure during a cycle.



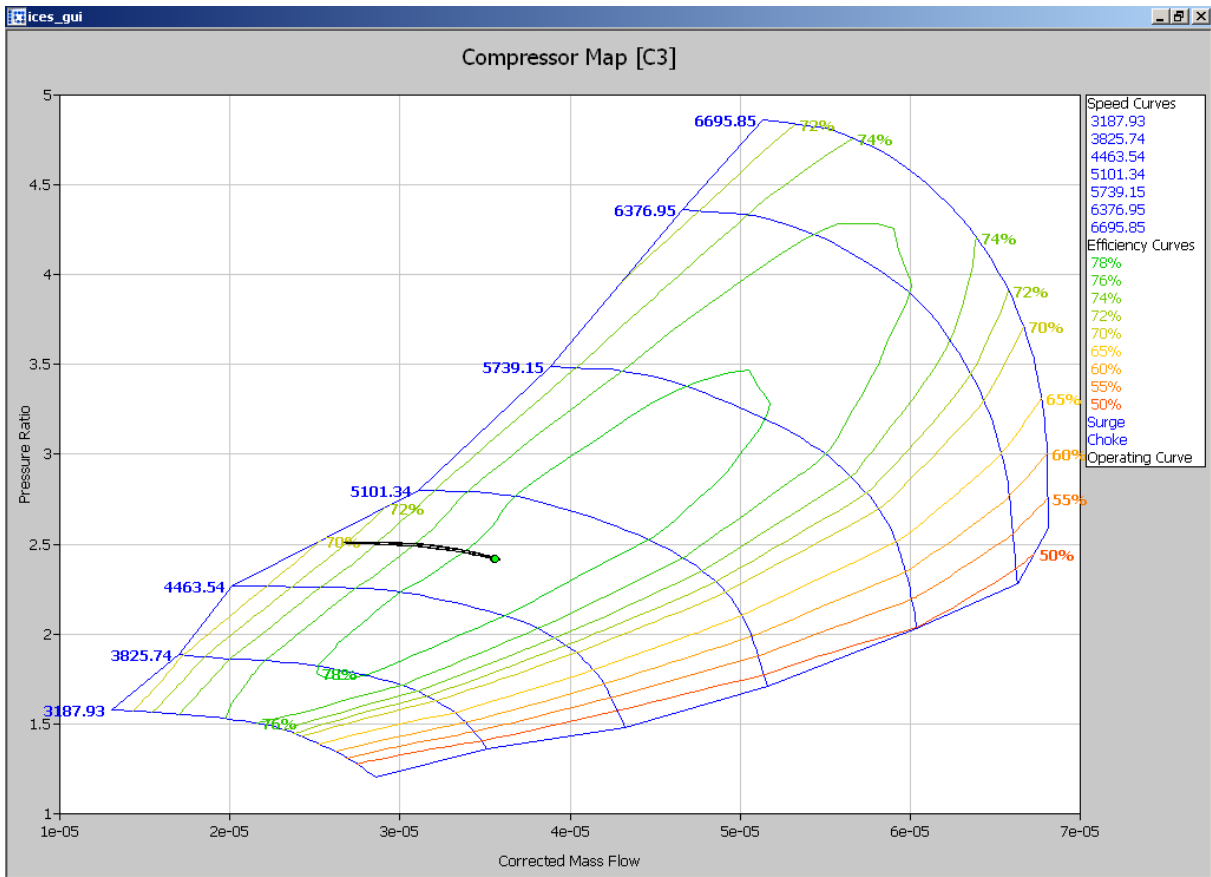
A plot showing the relationship between the cylinder temperature and pressure. These are the same data as the above diagram.



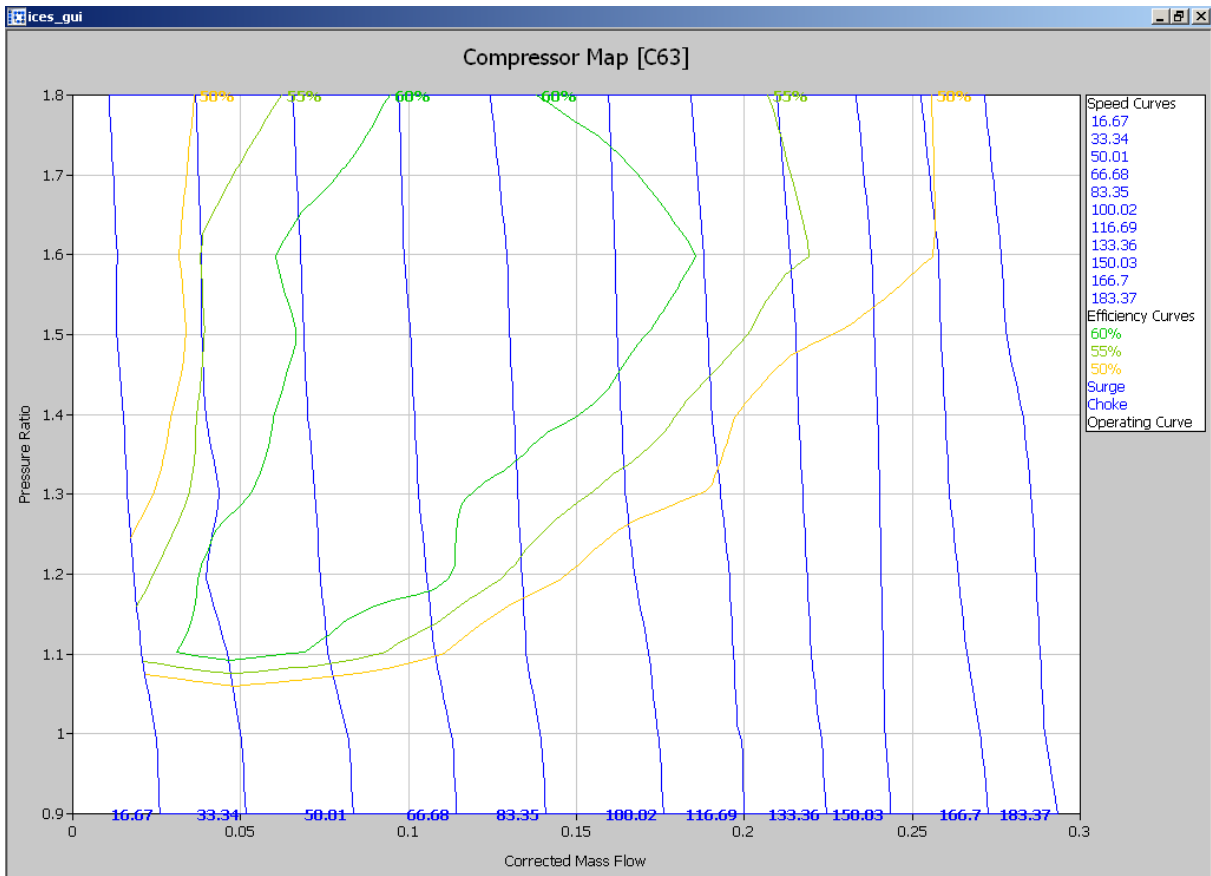
A rich plot showing 4 relevant cylinder parameters in a cycle: mass, temperature, pressure, and volume. The legend is shown in the top-left corner.



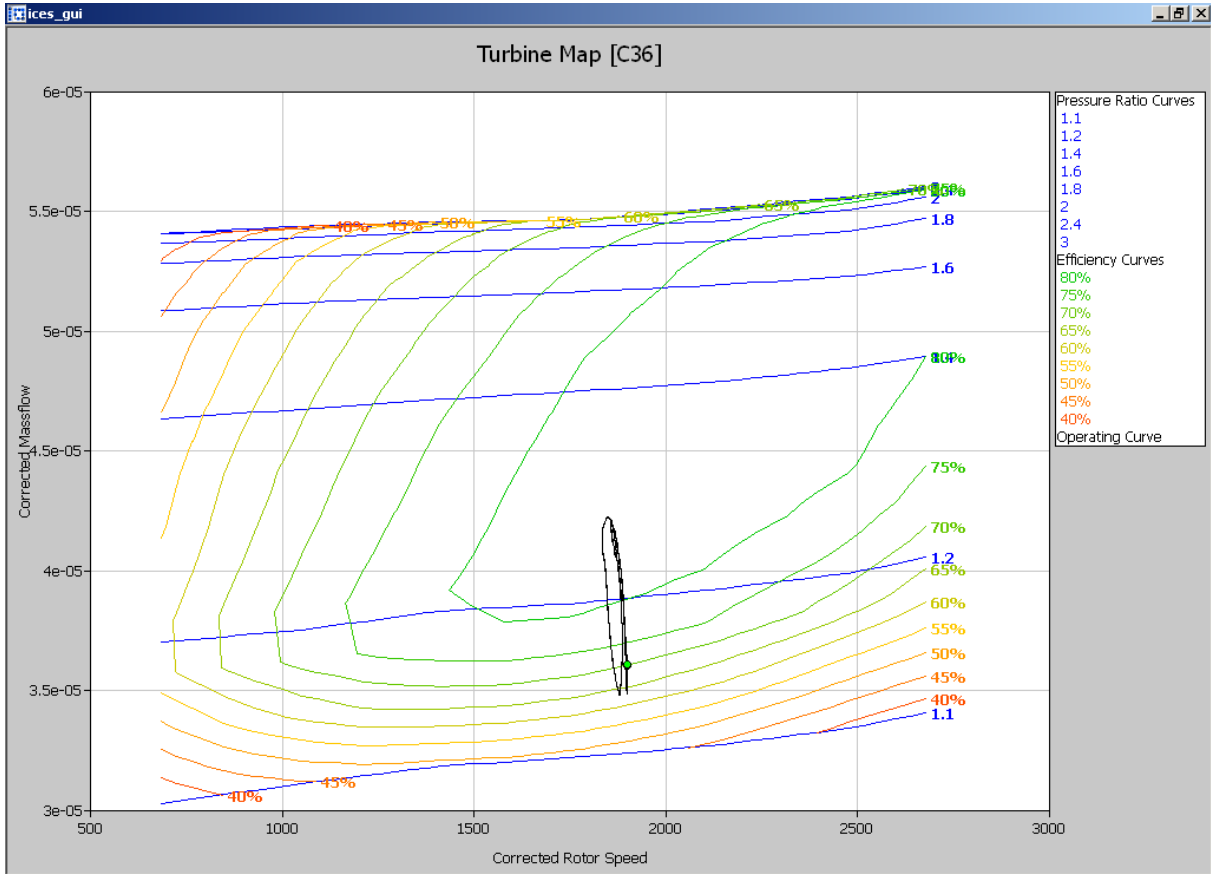
A plot showing temperature varying with the crank angle and with burned gas in a turbine volume.



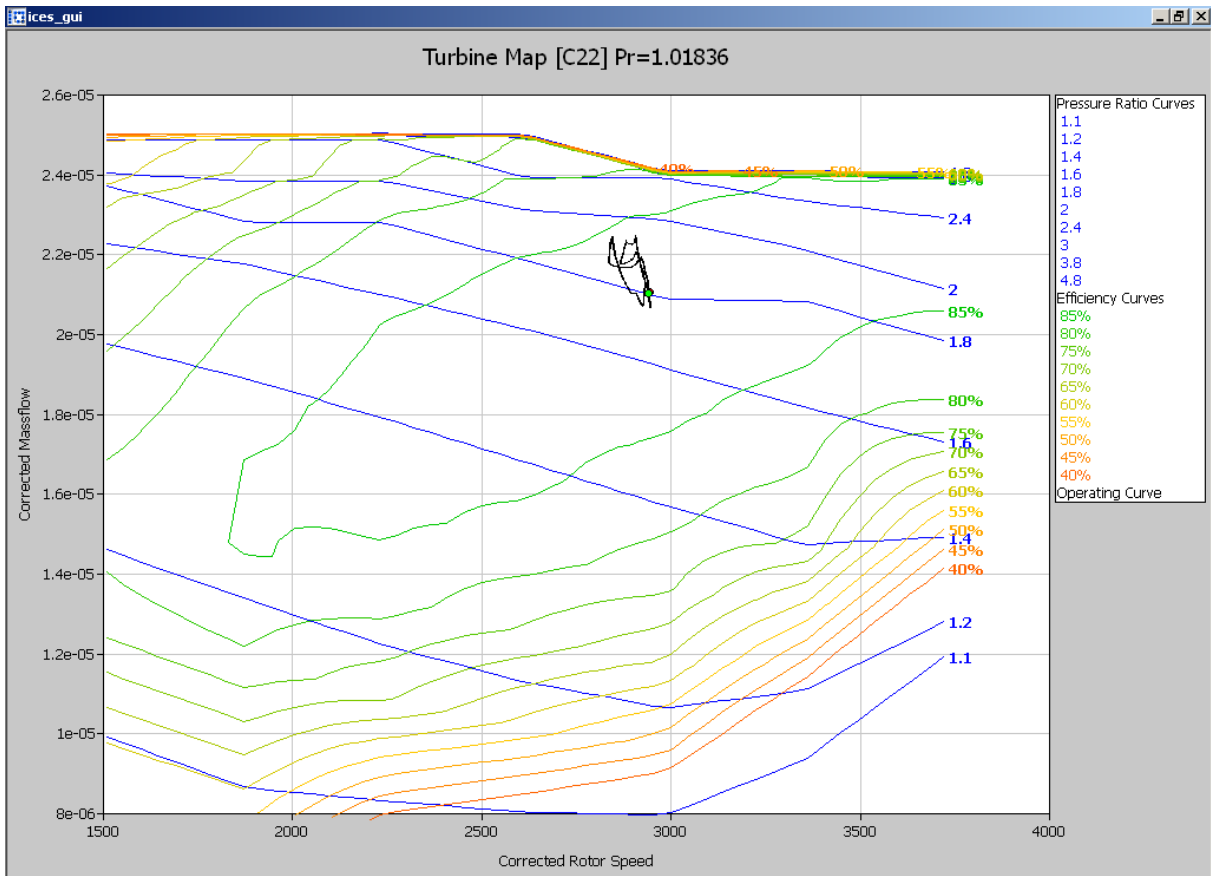
A plot showing a typical compressor map. Efficiency curves are colour coded. The black curve represents the operating curve in the current cycle. The green circle represents the starting point. The ending point is red, but it is covered by the starting point.



A plot showing a displacement compressor map.



A plot showing a single entry turbine map.



A plot showing a VGT turbine map.

6.8 Future Work

The post processor is, in its current state, useful for viewing output from ICES. However, there are additional features that can be implemented in order to help the users even further. Many applications today offer many ways of achieving the same goal. For example, when saving a file in a word processor, users can use the “file” menu, click the “save” button in the tool bar or press “Ctrl-S”. As of its current state, the plot view operations, such as item properties or changing plot mode, can only be accessed from a menu. Adding a tool box and keyboard shortcuts to the same actions will improve usability further.

The plot view can handle many curves. However, when dealing with multiple curves its often not obvious which curve belongs to which axes even if the axes colours varies. In order to solve this there is a need to visually connect the axes with each other within the view. In addition, when dealing with curves with few data points, there is sometimes a need to make the curve smoother. This can be solved by using cubical splines for instance.

It’s sometimes desired to make further calculations on the output with external applications such as Microsoft Excel. In its current state; the post processor only supports exporting of visual data such as printing, exporting to an image and the clipboard. A feature which allows the user to export the data to an external resource, such as a normal ASCII file, is often desired. Also, exporting the data itself to the clipboard in a convenient table structure might also be useful when working with Excel.

The last and most major issue for future work is a stand alone plotting application. As of now, the plotting tool can only be accessed within ICES GUI. However, the code has been generically structured and developing an external application using the existing components should not pose any difficulties. Many new concepts do however arise, such as importing of data. This can and should be doable in many ways and the imported data structure often varies, thereby making this a large project.

7 Conclusions

Even though the project hasn’t strictly followed any popular software development process, the use of an iteratively based workflow is very much preferable over the standard waterfall model. This is due to the fact that it’s almost impossible to perfect one phase before moving on to the next in one attempt. This project did indeed benefit from adding new and improving existing parts over several iterations. The reason for this is that it’s very hard to get a complete picture of all the classes and functions needed in order to fulfil the requirements. However, a solid fundamental design is preferable since this is a good method of getting a complete picture of the design. It also reduces future refactoring, thereby saving time.

QtDesigner has also proved to be a time saver. GUI programming can often be a tedious exercise since it often requires developers to write a small amount of code followed by a compile step to visually inspect the results. By using this tool it’s easier to get an overall picture and thereby making the process of selecting the appropriate widgets and layouts faster.

This project has implemented a few design patterns where applicable. As mentioned earlier, a future project is to create a stand-alone plotting application. This is an example of a good use of design patterns since these encapsulates the code and makes it easier to extract and implement the necessary parts for future software development.

Testing has been a large part of the project and has included many types of different tests. Even though most of the tests were done manually, the use of CppUnit and especially the use of QxTestRunner proved to be a very efficient tool for unit testing. This has assuredly helped the process of finding bugs. However, as with all unit testing there are limits as they do not catch all errors in a program since they only test the units or classes themselves. Integration and performance errors cannot be efficiently found using this method. Therefore, the use of integration and black-box testing provided an efficient method of finding these.

8 Dictionary

Crank angle – the position of the crank shaft in a combustion engine. In a 4-stroke engine a full cycle is composed of two revolutions, hence 720 degrees.

Cross-platform – a software implementation that's designed to work on multiple platforms such as Windows and Linux.

Delegate – a dynamic widget which renders itself. This is used in order to provide a custom editing interface for the developer.

Extrapolation – the process of constructing new data points outside a known set of data points.

Interpolation – the process of constructing new data points within a known set of data points.

Legend – an overview of the data series or curves present in a plot.

Signal/Slot system – the standard communication system used by Qt. Signals or events, such as a mouse click are attached to slots. The slot is a function which is to be implemented to provide an appropriate action for the signal.

VGT – Variable Geometry Turbine. A turbine with variable intake blades which regulate the intake pressure and thereby the rotation speed.

Widget – a graphical interface control which the user interacts with, such as a button or a window.

9 Sources

[1] *Qt Reference Manual*, commercial edition, version 4.2.3
<http://doc.trolltech.com/4.2>

[2] *C++ Library Reference*
<http://cplusplus.com/reference/>

[3] Sundin, Lars: *ICES Manual*, version 1.5-026

[4] Gamma, Erich: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, 1994

[5] Stroustrup, Bjarne: *The C++ Programming Language*, Addison-Wesley Professional, 1997

Appendix A

Requirements Specification

This document lists the minimum requirements for the post processor within ICES GUI.

Functional Requirements

R1: The tool must be able to 2D-plot ICES output parameters with the crank angle or with any other ICES output parameter.

R2: The tool must be able to plot all compressor and turbine map types supported by ICES.

R3: The tool must be able to plot any number of curves or maps within a plot.

R4: The tool must support a number of visual curve options. These are: colour, visual markers and dashed lines.

R5: The tool must support a number of visual axis options: These are: colour, location (top and bottom or left and right), minimum limit, maximum limit and minor steps.

R6: The tool must support the use of a grid, plot title and/or legend.

R7: The tool must support graphical exporting of the plot view to a file. File types are: PNG, JPG and BMP.

Non-functional Requirements

R8: The tool must be an extension of ICES GUI and must be accessible in this environment.

R9: The tool must be usable in regard to performance on a typical machine used today. A typical machine has the following specifications: Pentium 4/M 1.6GHz processor, ATI Radeon 7500 32MB graphics adapter, 512MB RAM and 10MB free disk space.

Appendix B

Time Plan

Planned and Actual times are in working days.

Iteration	Phase	Assignment	Planned time	Actual time
1	Basic Design Implementation		14	14
		Curve reading/creation	3	4
		Plot graphics	10	20
		Wizard	15	15
		Storage	2	1
	Func. Testing		1	1
2	Additional Design Implementation		10	8
		Map reading/creation	40	52
		Plot item options	4	3
	Func. Testing		1	1
3	Implementation			
		Plot item options	3	2
		Plot title/legend	5	3
	Mass Testing		2	2
Totals:			110	126

Appendix C

This section shows the part of the ICES GUI manual describing the graphical post processor.


VOLVO

Volvo Technology AB / Volvo Technology Corporation	Dokumentnamn/Name of document		Sida/Page
	INSTRUCTION		45 (69)
Utfördare (avd nr, namn, tfn, geo placering, sign)/Issuer (dept, name, phone, signature)	Datum/Date	Bilaga/Appendix	Reg nr/Reg. No
06180, Urban Flock, 031 – 322 67 54, CTP	2007-04-20 (DRAFT)	A-G	06180-05-10244-1
Ärende/Subject			
User Instruction for ICES Graphical User Interface – ICES GUI			

7.3.2 Viewing Curve Results and Compressor and Turbine Maps

All parameters that have been selected for storage as well as all compressor and turbine maps can be visualized and compared using the *Results Viewer* tool.

7.3.2.1 Results Viewer

To start the *Results Viewer*, right-click on the model and select *Results Viewer*, or click the *Results Viewer* quick button .

The data selecting part of the *Results Viewer* is composed of a 3-page wizard structure. The lower portion of the *Results Viewer* window contains the wizard controls. The pages can be browsed forwards by clicking on the *Next* button, or backwards by clicking on the *Back* button. At any page the *Results Viewer* can be cancelled by clicking on the *Cancel* button.

To display the selected results, the user can at any time press the *Finish* button which will then open the *Plot View*.

Figure 35 shows the wizard control buttons.



Figure 35: The wizard control buttons of the *Results Viewer*.

The wizard pages are: The *Data Selector*, the *Item Properties*, and finally the *General Properties*.

Data Selector

This page is used for selecting the desired data to plot. Figure 36 shows the *Data Selector* page. The tree to the left (*Model Data*) contains the parameters and maps that can be plotted from the model results. The right tree (*Plot Data*) lists the data that have been selected for the current plot.

VOLVO

Volvo Technology AB / Volvo Technology Corporation	Dokumentnamn/Name of document		Sida/Page
	INSTRUCTION		46 (69)
Utfärdare (avd nr, namn, ftn, geo placering, sign)/Issuer (dept, name, phone, signature)	Datum/Date	Bilaga/Appendix	Reg nr/Reg No
06180, Urban Flock, 031 – 322 67 54, CTP	2007-04-20 (DRAFT)	A-G	06180-05-10244-1
Ärendel/Subject			
User Instruction for ICES Graphical User Interface – ICES GUI			

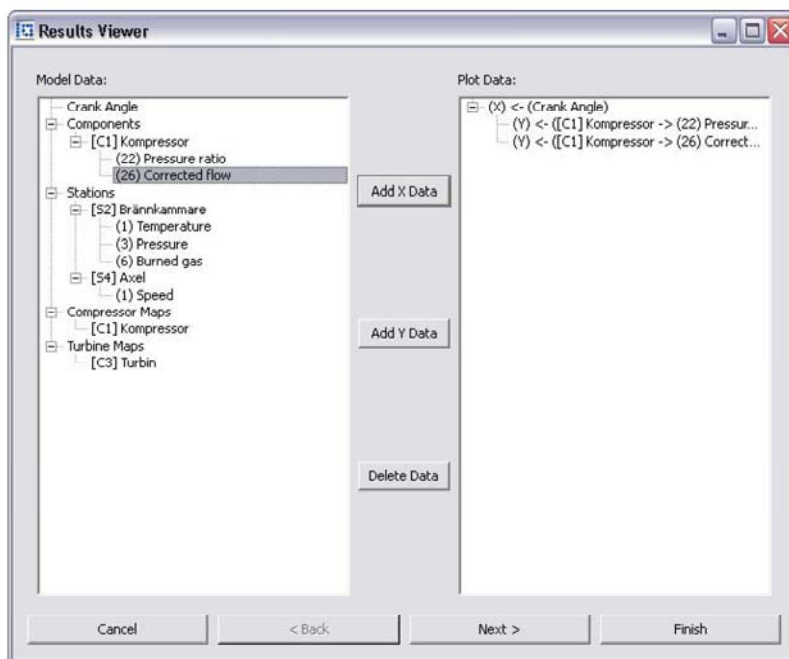


Figure 36: An example of the *Data Selector* page in the *Results Viewer*.

Items are added to the *Plot Data* tree by selecting them in the *Model Data* tree and clicking the desired *Add X Data* or *Add Y Data* button. Similarly, items are removed from the *Plot Data* tree by selecting them and clicking the *Delete Data* button.

Items in the *Plot Data* tree contain root items and leaf items. Root and leaf items can be added as x data, marked with (X) in the tree, or y data, marked with (Y). However, x data root items can only contain y data leaf items, and vice versa. This provides an interface where a root item can contain an arbitrary number of leaf items, giving support for combining several y axes to one x axis and vice versa.

Figure 37 shows an example of the *Plot Data* tree when having selected crank angle as x data and cylinder pressure and temperature as y data.



Figure 37: Plotting cylinder pressure and temperature against the crank angle.

To add a station or component parameter to the *Plot Data* tree:

1. Select the desired parameter from the *Components* or *Stations* categories in the *Model Data* tree.

VOLVO

Volvo Technology AB / Volvo Technology Corporation	Dokumentnamn/Name of document		Sida/Page
	INSTRUCTION		47 (69)
Utfärdare (avd nr, namn, itn, geo placering, sign)/Issuer (dept, name, phone, signature)	Datum/Date	Bilaga/Appendix	Reg nr/Reg No
06180, Urban Flock, 031 – 322 67 54, CTP	2007-04-20 (DRAFT)	A-G	06180-05-10244-1
Ärende/Subject			
User Instruction for ICES Graphical User Interface – ICES GUI			

2. Select the destination *root* item in the *Plot Data* tree (this is the item that the added item is plotted against) or select none to create a new root item.
3. Press the *Add X Data* to add the parameter as x data or press *Add Y Data* to add the parameter as y data.

Note: Selecting an X-root item will disable the *Add X Data* button and vice versa. Also, adding a new root item will automatically add a gray crank angle leaf item. This is for convenience, since plotting items against the crank angle is very common. Gray crank angle items act as normal crank angle items, however they are removed upon adding new leaf items.

To add a map to the *Plot Data* tree:

1. Select the desired map from the *Compressor Maps* or *Turbine Maps* categories in the *Model Data* tree.
2. Press the *Add Map* button.

Note: Maps can only be root items and cannot contain any leaf items.

To remove an item in the *Plot Data* tree:

1. Select the root or leaf item in the *Plot Data* tree.
2. Press the *Delete Data* button.

Note: Removing a root item also removes the corresponding leaf item(s).

Item Properties

The *Item Properties* page provides customization of the plot items generated by the data selected in the *Data Selector* page. The plot items are found in the *Plot Contents* tree to the left.

For station or component parameter curves the plot items are divided into 3 groups: curves, axes, and grids. For maps, the plot items are some different map properties. These have their own sections and contain their own plot items, as well as a *General* item which allows for general map modifications.

Figure 38 shows the *Item Properties* page.

VOLVO

Volvo Technology AB / Volvo Technology Corporation	Dokumentnamn/Name of document		Sida/Page
	INSTRUCTION		48 (69)
Utförare (avd nr, namn, fn, geo placering, sign)/Issuer (dept, name, phone, signature)	Datum/Date	Bilaga/Appendix	Reg nr/Reg No
06180, Urban Flock, 031 – 322 67 54, CTP	2007-04-20 (DRAFT)	A-G	06180-05-10244-1
Ärende/Subject			
User Instruction for ICES Graphical User Interface – ICES GUI			

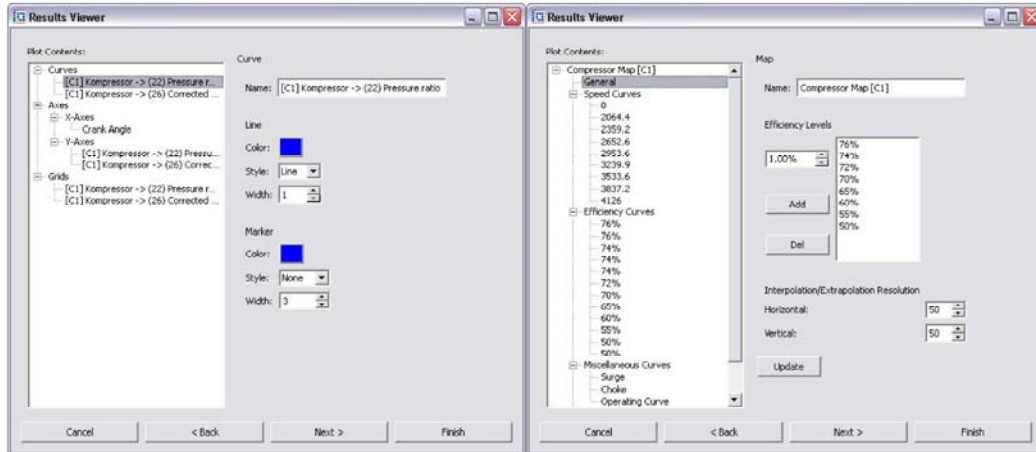


Figure 38: Examples of the *Item Properties* page showing items generated by curves (left) and by a compressor map (right).

Note: All properties available in the *Item Properties* page are available in the *Plot View* (the resulting graph/plot) as well.

To change the properties of a plot item:

1. Select the desired plot item in the *Plot Contents* tree. Corresponding properties controls will appear to the right.
2. Modify the desired properties.
3. Select a new plot item for modification in the *Plot Contents* tree, or press *Next* to save the new properties.

Note: When modifying a map *General* item, the *Update* button must be pressed in order to update the plot. This will discard all previous changes made to the corresponding plot items.

General Properties

The last page in the wizard provides general properties for the plot. Some examples:

- The plot title can be edited and shown on demand on different locations in the *Plot View*.
- The legend (a summary of all the curves in the plot) can be shown on demand on different locations in the *Plot View*.
- The plot background colors can be modified.

Figure 39 shows the *General Properties* page.

VOLVO

Volvo Technology AB / Volvo Technology Corporation	Dokumentnamn/Name of document		Sida/Page
	INSTRUCTION		49 (69)
Utfärdare (avd nr, namn, ftn, geo placering, sign)/Issuer (dept, name, phone, signature)	Datum/Date	Bilaga/Appendix	Reg nr/Reg No
06180, Urban Flock, 031 – 322 67 54, CTP	2007-04-20 (DRAFT)	A-G	06180-05-10244-1
Ärende/Subject			
User Instruction for ICES Graphical User Interface – ICES GUI			

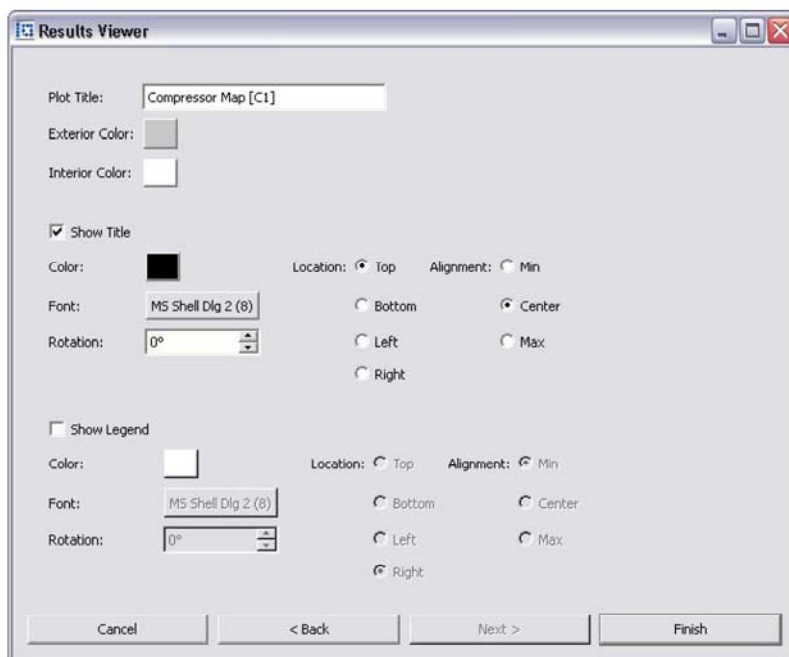


Figure 39: The *General Properties* page.

Note: All properties available in the *General Properties* page are available in the *Plot View* as well.

7.3.2.2. Plot View

The *Plot View* is the resulting view of the selections and properties made in the wizard. This view includes a plot area containing all the plot items, a title (optional), and legends (optional). The view is automatically updated to fit its contents when resizing the window.

Two modes can be selected while working with the *Plot View*; *Selection Mode* (standard) and *Measure Mode*. The *Selection Mode* allows for selection and modification of the plot items and the general plot properties. The *Measure Mode* allows for rough measurement within the plot area.

Figure 40 displays the *Plot View* in *Selection Mode*, where the right-click menu is shown.

VOLVO

Volvo Technology AB / Volvo Technology Corporation	Dokumentnamn/Name of document		Sida/Page
	INSTRUCTION		50 (69)
Utfärdare (avd nr, namn, ftn, geo placering, sign)/Issuer (dept, name, phone, signature)	Datum/Date	Bilaga/Appendix	Reg nr/Reg No
06180, Urban Flock, 031 – 322 67 54, CTP	2007-04-20 (DRAFT)	A-G	06180-05-10244-1
Ärende/Subject			
User Instruction for ICES Graphical User Interface – ICES GUI			

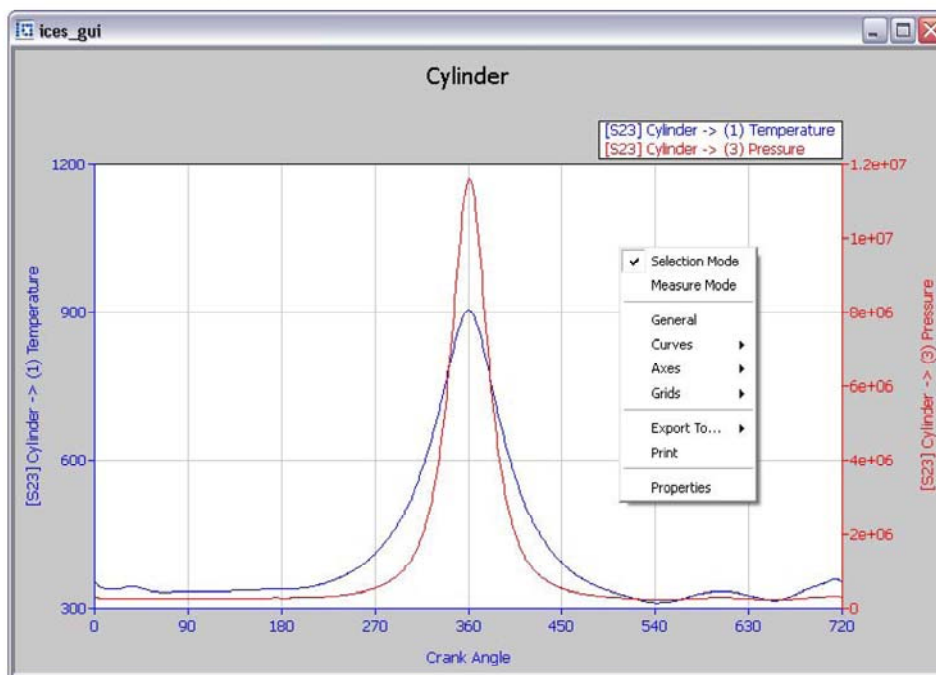


Figure 40: An example of the *Plot View* (in *Selection Mode*) showing the right-click menu.

To change mode:

1. Right-click anywhere within the *Plot View*.
2. Select *Selection Mode* or *Measure Mode* from the popup menu. The current mode is marked with a check box.

In Selection Mode

While in *Selection Mode*, modifications of the plot can be made. Left-clicking on an item will select it and left-clicking on an empty area will deselect the current item.

To modify an item:

1. Double-click on the item, right-click on the item and select *Properties*, or right-click anywhere in the view and select the item in the sub menus (*Curves*, *Axes*, or *Grids*). The corresponding properties window will appear.
2. Modify the desired properties and press *OK*. Pressing *Cancel* will undo the changes.

Note: Hidden or invisible items, such as curves not showing its lines or markers, can be modified by right-clicking on the plot area and selecting it from the sub menus.

DRAFT VERSION – WORKING DOCUMENT

VOLVO

Volvo Technology AB / Volvo Technology Corporation	Dokumentnamn/Name of document		Sida/Page
	INSTRUCTION		51 (69)
Uttfärdare (avd nr, namn, ftn, geo placering, sign)/Issuer (dept, name, phone, signature)	Datum/Date	Bilaga/Appendix	Reg nr/Reg No
06180, Urban Flock, 031 – 322 67 54, CTP	2007-04-20 (DRAFT)	A-G	06180-05-10244-1
Ärende/Subject			
User Instruction for ICES Graphical User Interface – ICES GUI			

In Measure Mode

While in *Measure* Mode rough measurements can be made in the plot area. This can be useful for example to make quick estimations of maximum or minimum values.

To measure in the plot area:

1. Put the mouse inside the plot area. A cross mouse icon will indicate this.
2. Move the mouse to the desired measure location and press and hold the left mouse button. The labels on the axes will hide and the pressed location will be displayed with a new label.
3. Release the left mouse button to end the measurement.

Note: Precision of the measurement is limited by the resolution of the plot area and should not be used for exact calculations.