Development of software package for event driven execution of multivariate models

Tomas Johnsson
Abstract

Development of software package for event driven execution of multivariate models

Tomas Johnsson

The BoardModel™ software system is today used as a visualization of, for example, logging of parameters in production and real-time predictions of responses such as formaldehyde emission or moisture content. The system is time based and consists of four main programs, the BMDC (saves and sends the incoming values), the View (shows the result to the screen), the Server (calculates the result) and the HDB exporter (export values to a text file).

This project aims at doing BoardModel™ event based and implement a new interface where the results can be shown. The need of the Server and the View programs in offline applications will be unnecessary, this will make the whole system much easier to use.

To make the system event based, SIMCA-QP from Umetrics AB will be used as calculating engine. An interface in C code which communicates with SIMCA-QP will be made. All other changes to the program will be made in C++.

The final version of the new BoardModel™ is event based, has support for multiple models and multiple y variables. The system also has the opportunity to send the calculated results as OPC. The new BoardModel™ consists only of BMDC with an inbuilt exporter and a new interface where the results are shown.
**Sammandrag**

BoardModel™ är ett mjukvarusystem som används för att visa till exempel värden av parametrar i produktionen och realtidsprediktering av bland annat formaldehyd och fukthalt. BoardModel™ är tidsbaserad och består av fyra olika program, BMDC (sparar och skickar vidare värden som kommer in), View (där resultaten visas), Server (som räknar ut resultaten) och HDB exporter (exporterar ut värden till en textfil).

Målet med detta examensarbete är att gör BoardModel™ häändelsestyrt och implementera ett nytt gränssnitt där resultatet kan visas. I och med detta kommer behovet av ett View- och ett Serverprogram att försvinna i offline applikationer och systemet kommer överlag att bli lättare att använda.

För att BoardModel™ ska bli häändelsestyrt kommer SIMCA-QP från Umetrics AB att användas som beräknings motor. För att kunna kommunicera med SIMCA-QP kommer ett C gränssnitt att byggas och resterande ändringar av programmet kommer att göras i C++.

Den färdiga versionen av BoardModel™ är häändelsestyrdd och innehåller stöd för flera modeller och fler y-variabler. Man kan också välja att skicka resultaten med hjälp av OPC. Den nya versionen består bara av BMDC med en inbyggd HDB exporter och ett nytt gränssnitt där resultaten visas.
# Table of Contents

1 Introduction ......................................................................................................................5
   1.1 Background..................................................................................................................5
   1.2 Goal .........................................................................................................................6

2 Theory ................................................................................................................................6
   2.1 Board Model™ .........................................................................................................6
      2.1.1 BMDC ..............................................................................................................7
      2.1.2 Server .............................................................................................................9
      2.1.3 View .............................................................................................................9
      2.1.4 HDB exporter ...............................................................................................10
      2.1.5 Relations in BoardModel™ ........................................................................11
   2.2 SIMCA-QP ..............................................................................................................12
   2.3 OPC ......................................................................................................................13

3 Material and setup ..........................................................................................................13
   3.1 BoardModel™ programming package ..................................................................13
   3.2 SIMCA-QP program package ..............................................................................13
   3.3 Visual Studio .......................................................................................................14

4 Implementation ...............................................................................................................14
   4.1 SIMCA-QP ............................................................................................................15
   4.2 Result view ...........................................................................................................15
   4.3 Multiple models and y variables .........................................................................18
      4.3.1 Multiple models .........................................................................................18
      4.3.2 Multiple y variables ..................................................................................19
   4.4 OPC ....................................................................................................................21
   4.5 HDB exporter ......................................................................................................22
   4.6 New HDB ............................................................................................................23
   4.7 BMDC window ....................................................................................................24
   4.8 Mean value ..........................................................................................................25
   4.9 Backup ................................................................................................................27
   4.10 Installation ..........................................................................................................28

5 Result ................................................................................................................................29
   5.1 Relations after the changes ...............................................................................30
   5.2 Testing ..................................................................................................................31

6 Discussion ....................................................................................................................32
   6.1 Future work .........................................................................................................32

7 Acknowledgments .........................................................................................................33

8 References ...................................................................................................................34

Appendix A ......................................................................................................................35
Abbreviation index

BMDC – BoardModel™ Data Collector
dll – dynamic link library
DModX – Distance to Model in X space
HDB – Historical Database
NIR – Near Infrared
OPC - Object Linking and Embedding (OLE) for Process Control
PLS – Partial Least Square
1 Introduction

1.1 Background

The BoardModel™ software system is used as visualization of, for example, logging of parameters in production and real-time predictions of responses such as formaldehyde emission or moisture content. Today the software is mostly used in the wood products industries for examples when manufacturing particleboard or biofuel pellets and then often used together with a NIR (Near Infrared) instrument [1]. The calculations are based on multivariate data analysis. The two methods used are PCA (Principal Component Analysis) and PLS (Partial Least Square).

The potential of BoardModel™ is very high and every line of business that uses or wants to use multivariate data analysis in their application is a possible user. BoardModel™ is mainly used in online applications, but has the potential to be used in a lab or similar as an offline application. It consists of four main programs:

- BMDC (BoardModel™ Data Collector) - collects data from different sources for example a NIR spectroscopy file or OPC signals from a process. The data is then stored in a database.
- BoardModel™ Server – a server application that handles the data that is stored in the database and makes the calculations using multivariate models made in SIMCA-P from Umetrics AB.
- BoardModel™ View - an interface to present the result from the Server. The results can then be viewed in real-time as lists, plots, trends etc. As BoardModel™ is a client/server application; a number of views can be used.
- HDB Exporter – a program who exports the data from the database to a text file. The text file can then be used for model calculation or other similar tasks, in for example SIMCA-P.
The whole software system is time based and independent of if a new value or a new observation has entered the system. This might cause some problems, for example that the software is calculating when it does not have to and results in empty rows in the View result list.

In this thesis work the task was to modify the BoardModel™ and mainly BMDC to be event based and more easy to use. This will make it more accessible and more likely to be installed in offline applications such as in labs, where new data appears on irregular bases. Also included in the task was to implement some new functions, integrate the HDB exporter in the BMDC and fix a number of bugs that the current version of BoardModel™ has. To make the system event based SIMCA-QP from Umetrics AB will be used as calculation engine.

1.2 Goal

The main goal of the project is to make the whole software system event based and at the same time make the program more user-friendly. A new interface where the calculated result is presented will also be made. The new software system should also have support for multiple models, multiple y variables and an opportunity to send the values with OPC. With help from the event based system and the new interface the need for the Server and the View in offline applications will disappear. The new event based software system should still be compatible with the View and the Server so it can be used in online applications where these two programs still are needed.

2 Theory

2.1 Board Model™

The BoardModel™ software system, consist of sub programs and described here is how they interact with each other.
2.1.1 BMDC

BMDC BoardModel™ data collector (BMDC) is a time based software that searches for new values (in this report it means a new NIR spectrum file, if not otherwise stated) at specific time intervals. These time intervals are set by the user (see Figure 1). When the program finds a new NIR file it triggers a set of events. First it checks if the new NIR file is identical with the old one, it also checks if the new NIR file really is new. Then it saves the spectrum together with a time stamp in a database (from here on called HDB (Historical Database)). The program also prints the spectrum and time stamp to the screen so the user can see it. An HDB is a binary file impossible to open with common programs.

![Figure 1](image.png)

**Figure 1** Dialogue to set the times when BMDC checks for a new NIR file and writes to the database.

BMDC have different settings that the user needs to configure before it can run for the first time. A path to a product file needed for the Server and the View must be set and the user must also specify the path to the HDB. If some of the paths do not exist the program will create them. Then the user must define which x variables the HDB will consist of in the tag management, under the settings menu. There the user can set what types of values BMDC will trigger on (a NIR file in our case) and then how many NIR tags to use, the name of each tag and the path to the NIR file (see Figure 3). A tag represents for example a value in the NIR spectra or an OPC value. After the mandatory settings the program also have a couple of optional settings, for example how often the program should make a backup of the HDB, in which interval BMDC should search for a new file and what information should be stored in the log file. The BMDC also has a window to show the different tag values that are stored in the HDB (HDB view) and a window to see
logging information (Message Log). Each window comes with the opportunity to change size and position of the window as well as size of the text and the number of rows with text in each window. All settings are stored in an ini file or in a serialization file. In Figure 2 a complete BMDC window with the HDB view and Message Log is shown.

![Figure 2](image1.png)

Figure 2 The old BMDC window, with the HDB view and Log view.

![Figure 3](image2.png)

Figure 3 Tag management window and the NIR settings dialogue.
2.1.2 Server

The Server is timed based and calculates the results from the data stored in the HDB and sends it to a result database. If no new observations are in the HDB the Server has nothing new to calculate and it will just print an empty value in the result database.

The Server is running as a service and has an icon in the activity field where the users are able to configure, start and stop it. The only changes that can be made directly in the Server program are the path to the result database and to the ini file. The rest of the settings are configured in a separate file in the windows directory or in the View program (see 2.1.3). To able to run the Server a path to the HDB must be set in the file in the windows directory. The time interval which the Server should calculate new values are set in the View. When a HDB exist and all settings are configured the Server is ready to run.

2.1.3 View

In the View the user specifies which model the Server will use to calculate the result with as well as many different parameters for example target values, what parameters that should be calculated and times between each execution. The View is where the result of the calculations, both as list with values and as plots, will be shown (see Figure 4). Layouts of the user interface are stored in a separate file called workspace.

At each time interval a new value is sent from the Server to the result database and then the View reads the database and print out the new value as a new line in the result list. If there is no value in the result database for that time interval the view just prints out an empty row. This can result in many empty rows if the BMDC does not send new values to the HDB as often as the interval time set in the View.
2.1.4 HDB exporter

The HDB exporter is a standalone program which is independent of the BMDC, View and Server, the only thing it need is an HDB. The user can choose which HDB to export and also set which observations, just specify between which times the observations should be. Then the user chooses where the text file with the exported values should be placed (see Figure 5). The text file, with the exported values, are ready for analysis in SIMCA-P or other programs to make and change model files or just take a closer look at the different x variables.
2.1.5 Relations in BoardModel™

In Figure 6, the relations in respect of data flow between the different programs and files in the BoardModel™ software package are shown. The process starts with a NIR file (or some other value like an OPC value) that comes from a NIR spectrometer. The BMDC searches for a new NIR file at a specific time interval and when it finds a new file it reads it and saves the spectra in an HDB. The Server, which always is in contact with the HDB, notices the new observation and reads it. With help of the model file, which is set by the user in the view, the result is calculated at specific time intervals and saved in a result database. Then the View reads the new value in the
result database and prints the value to the list and the plot diagram in the View workspace. If an export of the HDB is to be made, a separate program called HDB exporter is to be started, define which HDB to be exported and where to export it.

### 2.2 SIMCA-QP

SIMCA-QP is a calculation engine developed by Umetrics AB. It has no graphical interface and is implemented as a dynamic linked library (dll). The SIMCA-QP comes with two different interfaces to communicate, a C interface and a COM interface. In this case the C interface was used to make the calls to the dll file, as the whole BoardModel™ software is made in C++.

SIMCA-QP uses a model file to help with the calculations. The model files are created in SIMCA-P and consists of observations where the user know the results, in reality the x and y variables, which are used to make the calculations. In SIMCA-P the user has the opportunity to change the model and in the same time change many things in how SIMCA-QP calculates the results. For example the user can change scaling, how many models there are in each model file, which variables to be included in each model, what results to calculate and how to calculate them.

To make the calculations SIMCA-QP uses multivariate data analysis and more explicitly the PLS method to calculate the different results. SIMCA-QP has the opportunity to calculate many different results, but the two results that are interesting are the DModX and the y values. DModX is a value that shows how far from the model the observations are in the x space, the smaller value the better. A large DModX value is a sign of an observation outside the model range and should not be trusted. In these circumstances the best thing to do might be to make a new observation. Another thing that could result in a bad DModX value is if the model is not adjusted to this type of observation and a new model might be necessary. The y values that will be calculated and how to calculate it are also set in the model file.
2.3 OPC

OPC (OLE for Process Control or Object Linking and Embedding (OLE) for Process Control) is a standard for communication of real-time data between control devices. The idea behind OPC is to make a standard that all manufacturers of control devices could use. The advantage of this standard is that one OPC server from one manufacturer could be used with different OPC clients from different manufactures as long as it follows the OPC standard. Everyone can use OPC and everyone can develop an OPC server or client, no license is needed but the standard set by the OPC foundation has to be followed.[3]

3 Material and setup

3.1 BoardModel™ programming package

Of the three different main programs BoardModel™ consists of it is only the BMDC that will be changed and the code package consisted of the BMDC, some external dll files that BMDC uses and the code to make an installation program.

One problem that arose early was that the version of the BMDC that Casco Adhesives was using 3.61 and the code version was 4.00. The big difference was that the HDB did not match. As a consequence programs like the HDB exporter and the Server did not work with a HDB created with BMDC 4.0. That also means that the new version of the HDB could not work together with the Server and the View in an online application. Without the code to the Server or the HDB exporter it was impossible to change the code in the BMDC so that the HDB could work with the Server and HDB exporter or vice versa to change to code in the Server or HDB exporter to work with the new HDB. That is why the new program must be independent of the View and the Server.

3.2 SIMCA-QP program package
SIMCA-QP is a product from Umetrics AB and is shipped as an installation program that installs SIMCA-QP on the computer. The SIMCA-QP package includes some different help files to use during implementation. Included were documents that explain how it works as well as files that explain all the different functions. The software comes with many different code examples on how SIMCA-QP can be implemented. Finally it is shipped with a dll with the binary and some C header files to use when building the C interface. SIMCA-QP also needs a license file to run and this license file is specific for each computer.

### 3.3 Visual Studio

The C interface that communicates with SIMCA-QP, all changes and new functions in BMDC will be programmed in C/C++. The program used in this project was Microsoft Visual Studio 2005 and more specific Visual C++. The reason for this choice was that the current version of BMDC was made in Visual C++. Much of the programming already made and that will be made, will use MFC (Microsoft Foundation Classes). MFC is a set of predefined classes that Windows programming is built around [2].

### 4 Implementation

To get the system event based the Server and the View programs must be unnecessary. The Server will be replaced with SIMCA-QP as the calculating engine. The SIMCA-QP can be configured in many ways, in this case to be used to get the whole systems event based. With the Server out of the picture the View can also be replaced. Instead of the View program the BMDC will have a window built in, where the results and other useful information can be shown. To make the system user-friendly, the HDB exporter will be integrated into the BMDC program no external programs will be necessary.
4.1 SIMCA-QP

An interface is needed to communicate with SIMCA-QP, in this case a C interface. The interface is built up with functions that all have one thing in common, they have one input parameter that is a structure that holds information about which model file is used; the name of the variables set in tag management in BMDC, a project handle used to identify the project, the number of variables and the number of observations. The structure makes it easy to use these frequently used variables in all functions within the interface.

To run the SIMCA-QP, BMDC makes a call to the interface with an initiated structure (that is a structure with the default values), the tag names and the data from the NIR file as in-parameters. Then the interface uses the built in functions in SIMCA-QP to get information from the model to use when calculating, storing and showing the results to the screen. That information can consist of name and model number for each of the different models in the model file and name of the y and x variables in the model.

Before a prediction is made the interface checks if the variable names in the model and those defined in the tag management in BMDC correspond to each other. Then the interface uses the values for each variable name that corresponded and makes the prediction based on these values. With the prediction made, the C interface saves the predicted in a prediction handle. The prediction handle is then used to receive the result from the SIMCA-QP, in this case the DModX (Distance to Model in X space) and the predicted result, y. The results are then sent to BMDC to be shown on the screen and saved.

4.2 Result view

When SIMCA-QP has calculated the different results the interface sends them to the BMDC. Besides DModX and y BMDC also receives the model number, the names of the models and y variable names from SIMCA-QP. Then BMDC print the results to a text file where all the results are stored. In this way it will be possible to view old results and to see the results if for example
the computer or BMDC crashes. This text file is also part of the backup function (see 4.9). Finally the results are sent to a result view that is showed in the BMDC window.

In Figure 7 the result view is shown, it works like the HDB view or the log view. It will have one row for each observation. Each row has a number of columns, some of the columns are mandatory and some can be chosen by the user. The mandatory columns are:

- Time column – to show the time and date when the observation and prediction was made.
- Model Number – number of the model used for the prediction. This model number is set in the model file.
- ID – id of this observation. The user can choose to set the id of the observation manually, this is to know what the observation correspond to and to easier find the observation (see Figure 9 and Figure 10).
- DModX – calculated distance to model in x space.
- Y – predicted y value, the y column will also contain the name of the y variable.

![Figure 7](image)

Figure 7 The standard result view for the new BMDC.

The result view could also have a column for the mean value (see 4.8) and columns for each y in the model (see 4.3.2). The mean value can be optionally set in the same dialogue as the model file is set (see Figure 8). If a mean value will be calculated it will get a new column called mean (see 4.8 and Figure 20). The user can also set how many y variables there are in the model and will get one column for each y variable (see 4.3.2 and Figure 14).
Where the user choose y variables it is also possible to choose if any application specific id should be added or if the predefined value ‘ID’ should be used as the identity (see Figure 9). If the user wants to set the id a pop-up dialogue where the identity can be entered will be shown (see Figure 10).
The user also has the opportunity to change some of the settings of the window, a right click anywhere in the window will bring up the options menu where the user can change how many rows to be shown before the window starts to delete the oldest row. The text size and if the text should be shown as bold or not can also be set in the options menu (see Figure 11).

![Res View Options](image)

**Figure 11** Result view options dialogue, where the user sets number of lines and how the text should look like.

### 4.3 Multiple models and y variables

Some of the model files made in SIMCA-P could have more than one model and some of the models could have more than one y variable. To allow these types of model files some adjustments need to be made both in the SIMCA-QP interface and in the BMDC code.

#### 4.3.1 Multiple models

To be able to use more than one model in the same model file the interface needs to do all the different calculations for each of the models. The interface also needs to call the function that writes to the result file and the screen for each model. This was made by storing each different result from the SIMCA-QP in a matrix or a vector, and then it sends all the different results at the same time to the BMDC by looping over the number of models. All the results where then calculated and stored in matrixes and vectors, one vector for DModX, one matrix for y etc. In the matrix each column corresponds to one model and each row a value for that model. After looping all the models the interface sends the matrixes and vectors forth to the BMDC.
BMDC goes through each matrix column values and prints the results to the corresponding column in the result view and to the result file. When the matrix changes to the next column it also changes row in the result file and in the result view (see Figure 12). By implementing this method ensured that BMDC got all the results at the same time to be used with OPC (see 4.4) or other functions.

![Figure 12](image_url) The result view when an observation is made with seven different models in the same model file.

### 4.3.2 Multiple y variables

After the change to allow multiple models the SIMCA-QP interface did not needed any changes to allow multiple y variables because when the y results are stored in the matrix each row correspond to a y variable. To write the y results to the result file appeared to be straight forward. The task to be addressed in this case was when to write the results to the screen. The questions were, how should the result view change the number of columns, what should the names of the columns be and how does the y variable names from the SIMCA-QP get to the corresponding columns?

The solution in this case was to make a dialogue window where the user can set how many y variables there are in this model and the name of each y variable column (see Figure 13). The BMDC stores the settings in the ini file so it does not need to be set all the time. When the result view is made, it is made with the right numbers of y columns that correspond to the value set by the user. The names of each column are also chosen and set by the user. If no names are set, the
column header will just be empty. The name of the y variable in the model will also be printed out after the value in respective column and row. This feature was made to make it easier for the user to see which column that correspond to which y (see Figure 14).

![Set y variables](image1)

Figure 13 The dialogue where the number of y variables and the name of them are set. The names are separated with a comma to define which name each column has.

![Results from prediction](image2)

Figure 14 The result view when an observation is made with two y variables.

This solution also has some consequences. What will happen with the model files with many different numbers of y variables in the different models and what if the user sets the wrong number? For example if the model has three y variables but the user just set two? In the first case right results will be shown as long as the number of y variables is set to the highest value of the number of y variables in the models. When the number of y variables in the model is less then the number set by the user it will just print out a NaN (or a -99. value). In the second case only the first two values are shown and not the third value. The feature to show less y variables then what is in the model can be used if the model has more then one y variable but we are only interested in the first one.
A problem is that each time something about the numbers or names of the columns are changed, a new window needs to be opened to see the new changes. This also causes the results that were present in the view to disappear because a newly opened window always is empty.

4.4 OPC

BMDC have a built in OPC client and already has support for receiving OPC values. The user inserts OPC tags, chooses the OPC server and sets the variable names. This is done with the tag management in the same way as NIR tags are handled.

Now the BMDC should not just be able to receive values but also transmit the calculated y values to an OPC server. The first thing the user has to do to use OPC is to make an OPC tag in the tag management and there set the type and the name of the OPC value to send and which server to use (see Figure 15). The results are sent to the OPC server at the same time as the results are sent to the screen.

![Figure 15](image_url) The tag management view with an OPC tag and the OPC settings dialogue.

The problem here was to use multiple y variables and models at the same time. First the user must define a tag for each y variable in each of the models. Then BMDC needs to send all the
values to the OPC client at the same time or the second value will overwrite the first one. In the way the SIMCA-QP interface was made, all the results for one observation were sent at the same time. This made it possible to forward the results to the OPC, if there were more values then OPC tags only the first values will be used. If there are fewer values than OPC tags the tags that are not used will just have the same value as before.

The use of OPC values still have some problems; for example what will happen if the user wants to read and write values at the same time, how to distinguish between which tags that are used for reading and which for writing?

### 4.5 HDB exporter

In BMDC 4.0 the HDB did not work with the old HDB exporter. This means that the text file with the values exported from the HDB could not be made. The only way to make this feature work again was to make a new HDB exporter. This time the HDB exporter was implemented into the BMDC. The advantage is the ability to export a HDB while BMDC is running and an external program is not needed.

The new HDB exporter dialogue was made so it looked similar to the old layout so that present users would be familiar with the layout. The dialogue has a similar window and it works in a similar way. The Export HDB command was put under *File* in the menu. When the user wants to export a HDB and clicks the *Export HDB* command a dialogue shown in Figure 16 will appear. The time and data is automatically set to the entire HDB and the default pathways are set to the current active HDB and the last text file used as an HDB text file. The settings can be adjusted to export only the observations the user wants.
To make sure the new exported text file was compatible with SIMCA-P an old text file from BoardModel 3.61 was compared to the new one to make sure they look almost the same. All variable names at the beginning of the file followed by the observations starting with date and time and the values for the variables are separated by a tab. Making sure that the values and names were on the same tab was very important, otherwise SIMCA-P could not read it correctly.

4.6 New HDB

The HDB created by this new version of BMDC (BMDC 4.0) must have an identical layout as the old one in order to facilitate future incorporation of the new BMDC to the complete BoardModel™ software package (including Server and View programs). In this way the BMDC 4.0 could work both in offline and online applications where the Server and the View programs are needed. The big difference between the HDBs were that the headers did not correspond. This was solved by implementing a .cpp file included in the BoardModel™ code package, however not in the BMDC directory.

The new .cpp file was used and a new second HDB was made with it. When the Server used the new HDB, the Server was able to read it. By doing this another problem occurred, nothing in the new BMDC (BMDC 4.0) was compatible with the new .cpp file. The solution was to make two
HDBs when the Server and the View programs are needed; one for the regular work and one just for the Server.

Where the user sets the name of the HDB it is also possible to choose whether or not to use a server HDB (see Figure 17). When the ‘Use server HDB’ checkbox in BMDC is checked it will create two HDBs, one with the name the user has chosen and one with the same name plus “_server” to be used with the Server.

![Figure 17 The dialogue where to set the path to the HDB and whether to use a server HDB or not.](image)

### 4.7 BMDC window

Some bug fixing and small changes were made to the program in order to make it easier to work with BMDC.

Fast buttons to the different views and some symbols to respective button were made to make it easier for the user to bring up the different views. Essential bugs that were fixed were; one that made the views not loading in the right order and started the views minimized when starting the program. Another, were the times in the HDB view which appeared in the wrong order when loading an HDB.
The “About window” was updated (see Figure 18), and some changes were made both under the file and settings menus. The code to the ini file was changed which resulted in more settings saved and they were not reset every time the user started the program.

![Figure 18 The new about BMDC dialogue together with the new fast buttons to the views.](image)

### 4.8 Mean value

In the BoardModel™ View the user had the opportunity to choose whether to use the value calculated for each observation or a moving average over a fix number of observations. This feature is something that also should be implemented in the new BMDC (BMDC 5.0).

This is implemented as an optional feature where the user can choose whether or not to calculate the mean value and from how many values. When the checkbox is checked a new column in the result view called Mean will appear (see Figure 19 and Figure 20).
The mean value is then calculated in the same place as where the BMDC writes values to the result files. The result file that corresponds to the model is read and to calculate the mean the last number of y values are used (the number that is set by the user). For example if the user has set to calculate the mean with five values it takes the y values in the five last rows in the result file and calculates the mean value of them. If the model has more the one y variable separate mean value calculations are made one for each of the y variables and if the model file has more than one model, readouts from each model respective result file are made. Figure 20 shows how this can look in the result view.

Figure 19 The dialogue where the user change if the mean will be calculated or not and also sets how many values to calculate the mean with.

Figure 20 The result view when many observations with the same model have been made and calculations of the mean value for each case. How the mean value changes can be seen for each observation.
4.9 Backup

The backup feature did not work as intended in BMDC 4.0. Backup was made at the right time but without cleaning the running HDB from the backed up data. This needed a fix and at the same time a backup feature of the result files as well as the server HDB was implemented.

To activate the backup function the user must first go in to the *HDB and RES Backup* dialogue under *Settings* in the menu and check the activate checkbox. Here the user can set the interval wanted to make backups and also specify exactly when the next backup should happen (see Figure 21). BMDC then checks each time an observation is made if the time for backup has passed and if, makes a backup of the HDB. Then BMDC sets the time for next backup to as many days ahead as the interval set by the user. The backup of the result files will occur just before the result is printed out to file. The backed up files will be stored in a new directory named Backup and each file will be named with its original name plus date and time when backup was made (see Figure 22). The original file will then be cleaned and the new value will be printed to the file.

![HDB and RES Backup](image)

*Figure 21* The backup dialogue.
When the Server is running the server HDB cannot be cleared of the backup content. The Server must be stopped before cleaning the HDB. This is because two programs are accessing the same file which makes it impossible to change. It can be devastating in an online application to just close the Server so the user must decide when it is a good time to stop the Server to clean the server HDB.

### 4.10 Installation

The BMDC programming package also comes with the code to make an installation program for BMDC. The code was changed so it would include the new files created and also some files from the SIMCA-QP, so the user does not need to move them from the SIMCA-QP directory to the BMDC directory. A SIMCA-QP ini file is also included with the default path to the SIMCA-QP. When installing the program it will create one directory named HDB for the HDBs and one directory named Models for the model files (see Figure 23). The HDB directory will also be the default directory for the result files and the backup directory.
Figure 23 This is how the BoardModel directory will look like just after the installation.

5 Result

In Figure 24 the final BMDC window can be seen when everything is running as it should. The play button is pushed so the program searches for new NIR files and all three views are open and working. One of the views is the result view which shows the calculated values for a model with two y variables in this case (Perforator and ASTM(ppm)) and the mean column for the two y variables. The difference between an automatic id and an id set by the user can be seen. In the HDB view the NIR spectra for the four latest observations are shown. The user can scroll to the right to see the values for all the NIR tags as well as all other tags if the user for example has set temperature tags or OPC tags. At the bottom the log view is shown, here messages of what is happening will come up. The messages level, in this case, is set to error and critical so it only shows important message of what is happening.
Figure 24 The complete BMDC after a couple of observations have been made. We see the result view, HDB view and Log view at the same time.

5.1 Relations after the changes

Figure 25 How the BMDC and SIMCA-QP are related after the changes.

In Figure 25 a new relationship in respect of data flow between different parts of BoardModel™ can be seen, this relation diagram is only true for offline applications and in online applications...
that do not need the View. BMDC looks for a NIR file, just as before at a specific time interval which can be set as low as one second. The NIR spectrum is saved in the HDB and then sent to SIMCA-QP. This happens in BMDC and without the involvement of the HDB. The SIMCA-QP then calculates the results, with help of the model file set by the user. Results are then sent back to BMDC which then prints the result to the result file and to the result view on the screen. In the BMDC it is also possible to export a HDB to a text file, to be used in SIMCA-P, without needing to open another program.

5.2 Testing

To test the new BoardModel™ in a real environment the software was installed in the lab at Casco Adhesives. It was used in the every day work to analyze resin that was manufactured on the site. The resin sample was measured with a NIR spectrometer and the NIR file sent to BMDC and the result shown on the screen. This made the work for the lab assistant easier. Before, they needed to import the files to SIMCA-X and calculate the results there. It was time consuming and a very inefficient way of working. With the new BMDC they only needed to push the measure button to get the results automatically.

To get it tested in a different application the new BoardModel™ system was also successfully installed in another department of AkzoNobel, Expancel AB. There it was used both by personnel who had used the old BoardModel™ and by personnel who did not have any prior experience. The response was positive and the best thing was that BoardModel™ was very easy to use.
6 Discussion

With the help of SIMCA-QP the final version of BoardModel™ became event based. Together with the new result view and an inbuilt HDB exporter BoardModel™ has all the important functions to work without the Server and the View.

As an offline application where much of the functions of the Server and View programs are not needed, for example an application in a laboratory, it is working very well with all functionality for easy event based use.

In online applications there are often demands for plots and other functions from the Server and the View that the new BoardModel™ do not have. In cases when for instance plots are not needed the new BoardModel™ can be used in online applications too.

The entire BoardModel™ software package has a potential to be used outside of the particleboard industry, like it is used in Expancel AB. With this improved version of the program and the ability to easily get it going as an offline application, is something Casco Adhesives maybe should look into further.

6.1 Future work

The programming of software is never ending work; sometime it is necessary to say that this is enough. Besides all work that were done before this and things that were implemented, there is still much that can be done to make an even better BoardModel™.

The most important thing is to make a plot library to the BMDC to make BoardModel™ independent of both the Server and the View and to make BMDC work alone in online applications.

We should also think about implementing more results from the SIMCA-QP such as; the T matrix, root mean square error, probability matrix or cross validation. An opportunity to
choose which models in the model file to use for this observation could also be something that might be considered to implement.

There could also be a good idea to change how we choose y variables. Maybe the information should be taken from the model file, maybe the user should choose which y to use etc.

A closer look into the world of OPC should also be done. This is important to expand to BoardModel™ to other branches that do not only use a NIR file.

It is very important to listen to the final user, the one who is using BoardModel™ in the everyday work. What do the user need and what functions will work easier? Maybe it is where the continued focus about the BoardModel™ development should be.

7 Acknowledgments

First I would like to thank my supervisor, Morgan Grothage, for his great support and beautiful ideas through this degree project.

I would also like to thank my supervisor at Uppsala University, Olle Eriksson, for his guidance and for his willingness to take on this project.

Finally I would like to thank Casco Adhesives for giving me the opportunity to do this degree project. Many thanks also go to all the employees at Casco Adhesives in Sundsvall that made my time there so interesting and fun.
8 References


Appendix A

The code to the C interface that we use to communicate with SIMCA-QP.

```c
#include "stdafx2.h"
#include <string.h>
#include <stdlib.h>
#include "sqp.h"
#include <stdio.h>
#include "addResult.h"

int iNum = 0; // Counter for number of models

int SQP_Init(SQP* pObj, const char* szUSPName)
{
    #ifdef _WIN32
        pObj->mszUSPName = strdup(szUSPName);
    #else
        pObj->mszUSPName = (char*)strdup(szUSPName);
    #endif

    pObj->szFileBuffer = NULL;
    pObj->iNumCols = 0;
    pObj->iNumRows = 0;
    return -1;
}

void SQP_Destroy(SQP* pObj)
{
    free(pObj->mszUSPName);
    if (pObj->szFileBuffer)
    {
        SQX_ClearStringVector(&pObj->mstrVariableNames);
        free(pObj->szFileBuffer);
    }
}
```

void SQP_Run(SQP* pObj, FILE* pErr, SQX_StringVector* TagNameVec, SQX_StringVector* TagDataVec)
{
    int iNumVar;  /* Number of variables */
    int iModelNumber;  /* Model number */
    int iNumModels = 0;  /* Number of models in the project */
    int iIsFitted;  /* Variable to inform if the model is fitted or not. */
    int iModelIndex;  /* Model index */
    char* szErrString; /* String that will contain possible error. */
    char *szModelTitle;  /* The title for the model */
    int piColumnYSIZE;  /* The number of Y columns in the model */
    const char *nameString;  /* A string to store the tag names before we put them in the project */
    char *szColumnYName;  /* The name of the specified Y column */
    unsigned columns = 0;
    unsigned i;
    int iNumObs = 1;  /* Number of observations to predict. Here we always predict one observation at a time because we get one nir file at a time */

    SQX_StringVector oXVarName;  // Names of the x variables in a model
    SQX_FloatMatrix* pObsRawData = NULL;  // The raw data that will be used to do prediction on.
    SQX_ModelType eModelType;  /* A structure that will be send to Predict() containing the data that will be used for prediction. */
    SQP_ObservationRawData* pObservationRawData = NULL;

    // Set the variables in pObj
    columns = SQX_GetNumStringsInVector(TagNameVec);  // Columns is the number of variables in the nir file
    pObj->iNumCols = columns;
    pObj->iNumRows = iNumObs;

    // Add the project to a project hadle
    if (!SQX_AddProject(pObj->mszUSPName, 1, NULL /* Not a password protected project*/), &pObj->mszProjID))
    {
        szErrString = "Could not open project. AddProject failed."
        goto error_exit;
    }

    // Get number of models in the project
    if (!SQX_GetNumberOfModels(pObj->mszProjID, &iNumModels))
    {
        szErrString = "GetNumberOfModels failed."
        goto error_exit;
    }

    // Init title matrix to store the titles from all models
    SQX_InitStringVector(&strTitleVec, iNumModels);

    // Init Result matrix to store the results from all models
    SQX_InitFloatMatrix(&YResMatrix, 20, iNumModels);

    // Init DModX vector to store the DModX results from all models
}

error_exit:
SQX_InitFloatVector(&DModXVec, iNumModels);

// Init the model number vector to store the model number for all the models
SQX_InitIntVector(&vModelNumber, iNumModels);

// Init YVarName matrix to store the Y variable names from all models
SQX_InitStringMatrix(&YVarNameMatrix, 20, iNumModels);

// Get and store info for each one of the models.
for (iModelIndex = 1; iModelIndex <= iNumModels; ++iModelIndex)
{
    // Get the model number connected with this index.
    if (!SQX_GetModelNumber(pObj->mszProjID, iModelIndex, &iModelNumber))
    {
        szErrString = "GetModelNumber failed."
        goto error_exit;
    }

    // Check if so the model is of PLS type otherwise go to next model
    SQX_GetModelType(pObj->mszProjID, iModelNumber, &eModelType);
    if ((eModelType != SQX_PLS_Class) && (eModelType != SQX_PLS) &&
        (eModelType != SQX_PLS_DA))
    {
        SQX_SetDataInIntVector(&vModelNumber, iNum+1, iModelNumber);
        iNum++;
        continue; // Next model
    }

    // Check if the current model is fitted, the model must be fitted to get any data
    if (!SQX_IsModelFitted(pObj->mszProjID, iModelNumber, &iIsFitted))
    {
        szErrString = "IsModelFitted failed."
        goto error_exit;
    }

    // Only continue with this model if its fitted
    if (iIsFitted == 1)
    {
        // Get the model Title for this model and store it in a vector
        SQX_GetModelTitle(pObj->mszProjID, iModelNumber, &szModelTitle);
        SQX_SetStringInVector(&strTitleVec, iModelIndex, szModelTitle);

        // Get the variable names from the input vector and store it in the project
        SQX_GetColumnYSize(pObj->mszProjID, iModelNumber, &piColumnYSize);
        SQX_InitStringVector(&pObj->mstrVariableNames, columns+piColumnYSize);
        for (i=1; i<=columns; i++)
        {
            SQX_GetStringFromVector(TagNameVec, i, &nameString);
            SQX_SetStringInVector(&pObj->mstrVariableNames, i, nameString);
        }
    }
}
/*Get the model Y variable names and store it both in the
variable vector (because we need all the x and y variables names to get the
result) and in a matrix*/
for (i=1; i<=piColumnYSize; i++)
{
    SQX_GetColumnYNameByIndex(pObj->mszProjID, iModelNumber, i, 1, &szColumnYName);
    SQX_SetStringInVector(&pObj->mstrVariableNames, columns+i, szColumnYName);
    SQX_SetStringInMatrix(&YVarNameMatrix, i, iModelIndex, szColumnYName);
}

//Get x variable names from the model and store it in a vector
if (!SQP_GetQuantitativeNamesForPredict(pObj->mszProjID, iModelNumber, 0, &oXVarName))
{
    szErrString = "GetQuantitativeNamesForPredict
failed.";
    goto error_exit;
}
else
{
    //Set the data to use for the Prediction.
    iNumVar = SQX_GetNumStringsInVector(&oXVarName);
    //Number of x variables in the model
    if (iNumVar > 0)
    {
        //Yes, there are variables in this model.
        pObsRawData = (SQX_FloatMatrix*)malloc(sizeof(SQX_FloatMatrix));
        //If we have a input vector to read from
        if (!GetQuantitativeData(pObj, &oXVarName, pErr, TagDataVec, *&pObsRawData))
        {
            SQX_SetDataInIntVector(&vModelNumber, iNum+1, iModelNumber);
            iNum++;
            SQX_ClearStringVector(&oXVarName);
            free(pObsRawData);
            pObsRawData = NULL;
            szErrString = "GetQuantitativeData failed."
            continue; //continue with the next model
        }
    }
}

//Store the observation data in the variable that is sent to S-QP
if (pObsRawData != NULL)
{
    pObservationRawData = (SQP_ObservationRawData*)malloc(sizeof(SQP_ObservationRawData));
    pObservationRawData->pObsRawData = pObsRawData;
    pObservationRawData->pObsNames = NULL;
pObservationRawData->pObsLagData = NULL;
pObservationRawData->pObsLagNames = NULL;
}

//Make the prediction
if (!SQP_DoPredictions(pObj, pErr, iModelNumber, pObservationRawData))
{
    szErrString = "DoPredictions failed."
    goto error_exit;
}

//Clean up the observation raw data
if (pObservationRawData->pObsRawData)
{
    SQX_ClearFloatMatrix(pObservationRawData->pObsRawData);
    free(pObsRawData);
    pObsRawData = NULL;
    pObservationRawData->pObsRawData = NULL;
}

if (pObservationRawData)
{
    free(pObservationRawData);
    pObservationRawData = NULL;
}

SQQ_RemoveProject(pObj->mszProjID);
return;

//If we get an error we go down here
error_exit:
if (pObsRawData)
{
    SQX_ClearFloatMatrix(pObsRawData);
    free(pObsRawData);
    pObsRawData = NULL;
}
fprintf(pErr, "%s
", szErrString);
SQQ_RemoveProject(pObj->mszProjID);
return;

//A functions that calls the SIMCA-Q to do the predictions with our observation
static int SQP_DoPredictions(SQP* pObj, FILE* pErr, int nModelNumber, SQP_ObservationRawData *pObservations)
{
    //Init the prediction handle
    char* szErrString;
    SQX_PredictionHandle pHandle = NULL;

    //Calling SIMCA-Q with the observation
    if (!SQP_Predict(pObj->mszProjID, nModelNumber, pObservations, NULL, 0, &pHandle))
    {
        szErrString = "Predict failed."
        goto error_exit;
    }
//Call to the function to get the predicted result
SQP_GetResults(pObj, pErr, pHandle, nModelNumber);

// Clean up the predictions
SQP_ReleaseHandle(pHandle);
return 1;
error_exit:
    fprintf(pErr, "%s\n", szErrString);
    return 0;
}

static void SQP_GetResults(SQP* pObj, FILE* pErr, SQX_PredictionHandle pHandle, int nModelNumber)
{
    char*             szErrString;
    int               iNumComp = 0; //Number of components
    SQX_FloatMatrix   oDModX;       //Matrix for the DModX results
    SQX_FloatMatrix   oY;           //Matrix for the Y results
    int               iNumModels;   //Number of models
    int               row;
    int               column;
    float             fDModX;
    float             yRes;

    //Get the total number of models
    SQX_GetNumberOfModels(pObj->mszProjID, &iNumModels);
    //Get ModelNumber and store in a vector
    SQX_SetDataInIntVector(&vModelNumber, iNum+1, nModelNumber);
    //Get the number of components for this model
    if (!SQX_GetModelNumberOfComponents(pObj->mszProjID, nModelNumber, &iNumComp))
    {
        szErrString = "GetModelNumberOfComponents failed."
        goto error_exit;
    }

    //Get DModX,
    //In parameters: project handle, modelnumber, components, normalized, modeling power weighted
    //Out parameter: DModX
    if (!SQP_GetPredictedDModX(pHandle, -1 /* model number, not used */, NULL /*All components*/, 1, 0, &oDModX))
    {
        szErrString = "GetPredictedDModX failed."
        goto error_exit;
    }
    else
    {
        //A for loop to store all the DModX result in a vector for all models
        row = SQX_GetNumRowsInFloatMatrix(&oDModX);
        for (column = 0; column < SQX_GetNumColumnsInFloatMatrix(&oDModX);
            ++column)
        {
if (!SQX_GetDataFromFloatMatrix(&oDModX, row, column+1, &fDModX))
{
    szErrString = "SQX_GetDataFromFloatMatrix failed."
    goto error_exit;
}
SQX_SetDataInFloatVector(&DModXVec, iNum+1, fDModX);
}

SQX_ClearFloatMatrix(&oDModX);

//Get Y
//In parameters: project handle, modelnumber, components, scale, back transform, column y indices
//Out parameter: Y
if (!SQP_GetPredictedY(pHandle, -1 /* model number, not used */,
                iNumComp, 1, 1, NULL /*All column*/, &oY))
{
    szErrString = "GetPredictedY failed."
    goto error_exit;
}
else
{
    //For loop to store the result in a new matrix for all the models
    for (column = 0; column < SQX_GetNumColumnsInFloatMatrix(&oY); ++column)
    {
        for (row = 0; row<SQX_GetNumRowsInFloatMatrix(&oY); ++row)
        {
            SQX_GetDataFromFloatMatrix(&oY, row+1, column+1, &yRes);
            SQX_SetDataInFloatMatrix(&YResMatrix, row+1, iNum+1, yRes);
        }
    }
}
SQX_ClearFloatMatrix(&oY);
iNum++; // count iNum up for the next model
//Print results on the screen
//When we got through all models iNum = iNumModels
if (iNum == iNumModels)
{
    //A call to addRes to print the result to the screen
    addRes(DModXVec, YResMatrix, vModelNumber, YVarNameMatrix, strTitleVec);
    SQX_ClearFloatMatrix(&YResMatrix); //Clear the res matrix to be redy for the next observation
    iNum=0; //set iNum back to 0 to be redy for the next observation
}
return;
error_exit:
    fprintf(pErr, "%s
", szErrString);
return;
```c
int GetQuantitativeData(SQP* pObj, SQX_StringVector *pstrVariableNames, FILE* pErr, SQX_StringVector* fValVec, SQX_FloatMatrix *pVariableMat)
{
    SQX_IntVector oIndexVector;
    int iIndexVectorSize;
    char *tmpBuffer;
    unsigned iCol=1, iRow=1;

    // A call to the function who checks so all variable names is the same in
    // the model and in the tag name vector.
    // Store the index in o IndexVector
    if (!GetIndexVectorFromStrings(pObj, pstrVariableNames, pErr, &oIndexVector))
    {
        return 0;
    }

    // Get the size of the index vector.
    iIndexVectorSize = SQX_GetIntVectorSize(&oIndexVector);

    // Get number of x variables in the model.
    if (SQX_GetNumStringsInVector(pstrVariableNames) < 1)
    {
        return 0;
    }

    // Check so there is at least one variable that matches
    if (iIndexVectorSize < 1)
    {
        fprintf(pErr, "Can not find the qualitative variable names in the\n        nir file.\n        ");
        return 0;
    }

    // Init a float vector to store the data in.
    if (!SQX_InitFloatMatrix(pVariableMat, pObj->iNumRows, iIndexVectorSize))
    {
        fprintf(pErr, "SQX_InitFloatMatrix failed.\n        ");
        return 0;
    }

    // Read one value at a time.
    for (iRow = 1; iRow <= pObj->iNumRows; iRow++)
    {
        int iIndex = 1;
        int iVectorIndex = 0;
        int iDataFromVector = 0;
        float fVal = 0;
        for (iCol = 1; iCol <= pObj->iNumCols; iCol++)
        {
            for (iVectorIndex = 1; iVectorIndex <= iIndexVectorSize;
                 ++iVectorIndex)
            {
                SQX_GetDataFromIntVector(&oIndexVector, iVectorIndex, &iDataFromVector);
                if (iIndex == iDataFromVector)
                {
                    // This is the value we want. Convert it from a
                    string to a float.
                    SQX_GetStringFromVector(fValVec, iDataFromVector, &tmpBuffer);
                }
            }
        }
    }
}
```
fVal = atof(tmpBuffer);
// and store it in a float matrix
if (!SQX_SetDataInFloatMatrix(pVariableMat, iRow, iVectorIndex, fVal))
{
    SQX_ClearIntVector(&oIndexVector);
    fprintf(pErr, "SQX_SetDataInFloatMatrix failed.\n");
    return 0;
}
break;
}
++iIndex;
}

SQX_ClearIntVector(&oIndexVector);
return 1;

int GetIndexVectorFromStrings(SQP* pObj, SQX_StringVector *pNames, FILE* pErr, SQX_IntVector *pIndexVector)
{
    const char *szVarName;
    const char *szStringIt;
    int iIndex;
    int iVarName=1;
    int iCheck = 0;
    int iRowCount=1;
    int init=0;

    // Go through all variable names. Count how many there are matching in file.
    for (iVarName = 1; iVarName <= SQX_GetNumStringsInVector(pNames); ++iVarName)
    {
        SQX_GetStringFromVector(pNames,iVarName, &szVarName);

        // Find this name in the project vector with variable names from the tag name input vector
        for (iIndex=1; iIndex <= SQX_GetNumStringsInVector(&pObj->mstrVariableNames); iIndex++)
        {
            SQX_GetStringFromVector(&pObj->mstrVariableNames, iIndex, &szStringIt);

            // Is the variable names from the projec name vector and model name vector the same
            if (strcmp(szStringIt, szVarName) == 0)
            {
                init++;
                // Its the same name, count init up
                break;
            }
        }
    }

    // Check if find any variables that matched
    if (init == 0)
    {
        fprintf(pErr, "Could not find any matching variables in file.\n");
    }
return 0;

// Init the return vector
if (!SQX_InitIntVector(pIndexVector, init))
{
    fprintf(pErr, "SQX_InitIntVector failed.
");
    return 0;
}

// Go through all variable names and store them in the out vector.
for (iVarName = 1; iVarName <= SQX_GetNumStringsInVector(pNames); ++iVarName)
{
    iCheck = 0;
    SQX_GetStringFromVector(pNames, iVarName, &szVarName);

    // Find this name in the project vector with variable names from
    // the tag name input vector
    for (iIndex=1; iIndex <= SQX_GetNumStringsInVector(&pObj->mstrVariableNames); iIndex++)
    {
        SQX_GetStringFromVector(&pObj->mstrVariableNames, iIndex, &szStringIt);
        if (strcmp(szStringIt, szVarName) == 0)
        {
            // This is the name, store the index of the variable in
            // the vector
            if (!SQX_SetDataInIntVector(pIndexVector, iRowCount, iIndex))
            {
                SQX_ClearIntVector(pIndexVector);
                fprintf(pErr, "SQX_SetDataInIntVector
failed.
");
                return 0;
            }
            iRowCount++;
            iCheck = 1;
            break;
        }
    }
    if (iCheck == 0)
    {
        // The string was not found.
        char strError[100];
        strcpy(strError, "Variable ");
        strcat(strError, szVarName);
        strcat(strError, " could not be found.");
        fprintf(pErr, "%s
", strError);
        break;
    }
}
if (iCheck == 0)
{
    SQX_ClearIntVector(pIndexVector);
    return 0;
}
return 1;