ISSN 0280–5316
ISRN LUTFD2/TFRT--5536--SE

Modelling and Object Oriented Implementation of a Windows Based PLC Configuration Tool

Henrik Petersson

Department of Automatic Control
Lund Institute of Technology
September 1995

Department of Automatic Control **Lund Institute of Technology** P.O. Box 118 S-221 00 Lund Sweden	*Document name* MASTER THESIS *Date of issue* September 1995 *Document Number* ISRN LUTFD2/TFRT--5536--SE
Author(s) Henrik Petersson	*Supervisor* Phillipe Pleche, ABB and K-E Årzén, LTH *Sponsoring organisation*

Title and subtitle

Modelling and Object Oriented Implementation of a Windows Based PLC Configuration Tool. (Modellering och objektorienterad implementation av ett Windows-baserat konfigurationssystem för PLC).

Abstract

The purpose of this master thesis is to model and develop a Windows application, designed to handle the configuration process of the ABB Procontic CS31 automation system. This program is labelled CS31CONFIG. Initially, it runs independently, but as a long term goal, it will serve as an integrated tool in a planned new version of the ABB Procontic CS31 programming software, under development by an external society. CS31CONFIG allows the user to specify necessary configuration parameters depending on the actual I/O system topology, and thereafter transmit this information to a master Programmable Logic Controller (PLC). The programming was carried out in an object oriented approach, and the code was written using the Microsoft Visual C++ compiler.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title 0280–5316		*ISBN*
Language English	*Number of pages* 31	*Recipient's notes*
Security classification		

Contents

Acknowledgements

I would like to thank my supervisor Philippe Pleche at ABB Control for his excellent guidance and help throughout this master thesis. I would also like to express my gratitude to the rest of the team CS31, especially Marc Jaffrennou and Claude Henrion for their valuable advice.

At the Department of Automatic Control at Lund Institute of Technology, I would like to thank Karl-Erik Årzén for supervising my work, making this master thesis possible.

Finally, Sebastien Noël at ABB Control has given me information and indispensable technical support regarding several software and computer systems. Thank you!

Lund, September 1995

Henrik Petersson

1. Introduction

The purpose of this master thesis is to model and develop a Windows application, designed to handle the configuration process of the ABB Procontic CS31 automation system. This program is labelled CS31CONFIG. Initially, it runs independently, but as a long term goal, it will serve as an integrated tool in a planned new version of the ABB Procontic CS31 programming software, under development by an external society. CS31CONFIG allows the user to specify necessary configuration parameters depending on the actual I/O system topology, and thereafter transmit this information to a master Programmable Logic Controller (PLC). The programming was carried out in an object oriented approach, and the code was written using the Microsoft Visual C++ compiler.

In chapter 2, the hardware as well as the programming software of the ABB Procontic CS31 are described. The problem is defined in chapter 3, and the goal of this master thesis is also stated. Chapter 4 presents the definition of the CS31 database, applicable to future PLCs. Chapter 5 explains why the application CS31CONFIG was written for Windows using C++, and it introduces the Microsoft Foundation Class Library - the tool used in the implementation. However, no attempt is made to explain object oriented programming, which is considered as a prerequisite. This chapter also describes Windows programming in general, and the fundamentals of an application under Windows. Chapter 6 describes the implementation of the application, by explaining the structure of CS31CONFIG and how it was modelled. The chapter also discusses the transmission and the interaction with the CS31 system. Chapter 7 is a summary, and finally, appendix A presents a complete description of the CS31 database.

2. ABB Procontic CS31

2.1 ABB Procontic CS31 Automation System

The ABB Procontic CS31 [1], commonly referred to as CS31, is an intelligent decentralised automation system, developed by ABB Control, Chassieu and ABB Schalt- und Steuerungstechnik (ABB SST), Heidelberg. It is mainly designed for small to medium size industrial processes. In general, the client handles the installation and the programming, while ABB offers technical support. The system is controlled by a master PLC, and in addition, up to 31 remote input/output units (I/O units) can be connected. The units are linked together via a simple pair of twisted wires, forming an automation network. The concept of using a simple pair of wires instead of conventional cabling, drastically reduces installation costs as well as it assures and simplifies modularity.

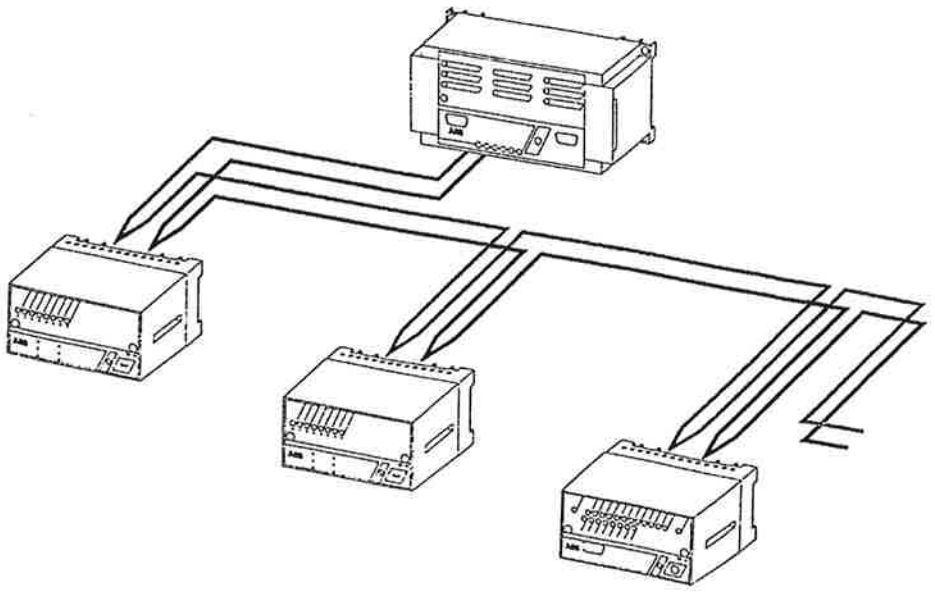


Figure 2.1 The ABB CS31 automation system; the PLC KR91 can control up to 31 I/O units.

2.2 Hardware

The master PLC executes a user provided program, i.e. instructions how to behave and evaluate information. It communicates with its environment by means of I/O units. Each I/O unit contains a number of I/O channels, analogue or binary, functioning as sensors and/or actuators. Thus, these I/O units can provide the master PLC with input data, for example a temperature measurement or the setting of a switch. This information can thereafter be evaluated in order to provoke a desired action. The action is carried out by the I/O units upon a command reception from the PLC, for example reduction of a radiator heating or toggling of a switch. In order to communicate via a simple connection, each unit is assigned an address and each channel a channel number. Any channel specific information transmitted in the network contains the address and the channel number of the command target, i.e. channel of receiving unit. Similarly, any unit specific information (commands concerning all channels of a unit) contains an address of the receiving unit. The I/O channels can be configured, altering their behaviour, and adapting them to an actual task. These different behaviours are described by a global set of configuration parameters. To each channel, a specific subset of the configuration parameters is applicable, corresponding to the particular hardware. Examples of these parameters are input filter time and cut wire detection for binary channels, and analogue measure range of tension/current.

2.3 Software

In order to manage and provide the CS31 PLCs with instructions, ABB SST has developed the software PC33 [2], running on Intel platforms under DOS. The operator works with PC33 to program solutions for different control problems, for example PID regulation etc. PC33 supports Function Block Diagram (FBD), Instruction List (IL) and Ladder Diagram (LD). In addition to PC33, all programming features are also accessible in terminal mode, which simply is an ASCII character interface between the PC and the PLC, following a CS31 syntax. In all, there exist about 50 commands to program and interact with the PLC. PC33 can be regarded as a graphical interface, using terminal mode commands or blocks of them. All communication is handled via a serial port operating at 9600 bps. Considering ABB's clients often limited technical know-how, PC33 tends to be far too complicated. To adapt the CS31 line of products to a growing market and to simplify the handling, ABB Control has engaged an external society to develop a new software under Windows, which is due for delivery by the end of 1995. This program will replace the PC33. In addition to the PC33 programming languages, it will also support Sequential Function Chart (SFC), Structured Text (ST) and C/C++ code. The user will work in a simple graphic environment, and for greater supervision, several tools will be available, for example debugging and simulation features. As earlier mentioned, CS31CONFIG will play an important part of this software.

3. Problem Definition

The configuration process of the CS31 serves to set the unit and channel associated configuration parameters at user provided values. This can be today be handled in terminal mode by the command MAIL [1]. While MAIL provides a perfectly well defined mean of configuration, it is rather tedious from a user perspective to type one MAIL for each channel, remembering that for instance the PLC KR91 can handle up to 992 binary channels. The PC33 implementation of MAIL is also quite awkward and time consuming. With a little knowledge of the command MAIL, a user can more rapidly configure a system by means of sending MAILs than using PC33. Clearly, this could be handled more efficiently. In addition, today's generation of PLCs are unaware of the actual settings of the parameters, since only the type of the remote unit and the corresponding address are stored in its memory. In order to intelligently respond to certain situations, future CS31 PLCs are planned to store all information regarding the configuration. This demands the definition and creation of a new configuration command, applicable to future PLCs as well as a supporting software. By creating a user provided database containing all relevant configuration information, storing it in the memory, the PLC will be able to handle the configuration process internally, and solve certain problems independently. When defining the structure of the database, there are certain technical constraints to consider. This concerns especially the compactness, and the simplicity of the decoding of the database, as well as flexibility towards future I/O units with unknown configuration parameters.

While improving the performance as the CS31 system becomes more complex, it is important that user interaction remains simple. This is a central part of the CS31CONFIG specification, and a major effort has been spent to attain user simplicity.

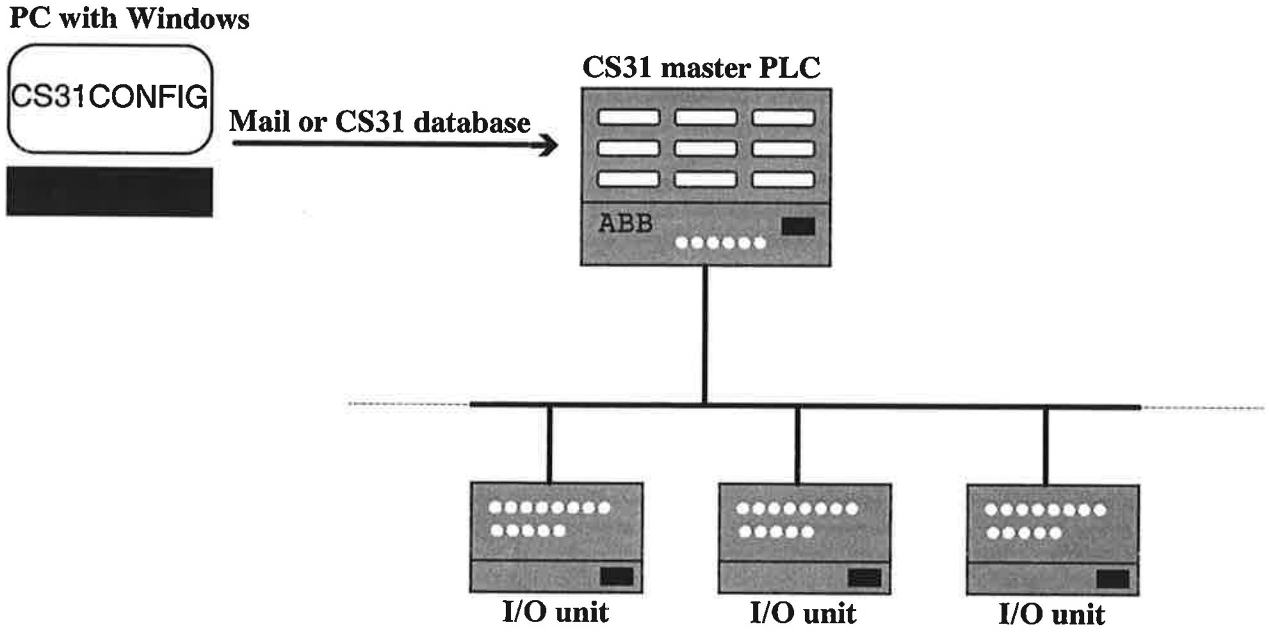


Figure 3.1 The relation between the CS31 system and its supporting software. The latter contains for instance programming, configuration and diagnostic features.

To summarise, the topic of this master thesis is the development of CS31CONFIG, a configuration tool which allows a user to easily enter relevant settings of the configuration parameters. Depending on the actual PLC, this information will be translated into either MAIL commands, assuring compatibility with existing PLCs, or a database format, applicable to the next generation of PLCs. Finally, CS31CONFIG will transmit the information via a serial port to the PLC. CS31CONFIG replaces the existing PC33 configuration implementation. As a second part of the initial problem definition was a diagnostic tool designed to run in close co-operation with CS31CONFIG, exchanging information. The main purpose of this program is to evaluate error information stored in the PLC, and provide the user with means for solving them. The master thesis has not attained this part of the specification due to unexpected difficulties during the modelling phase. Also, the information flow between the two tools was not defined in the beginning of the project, significantly delaying the development.

4. Definition of CS31 Database

4.1 Layout

The database contains information about the actual setting of the configuration parameters. These can be regrouped into unit and channel specific parameters. The global set of configuration parameters shown in table 4.1, is the information required by MAIL in order to configure the CS31 system.

Parameter	**Possible Settings**
Unit category	BI, AI, BO, AO, BIO, AIO[1]
Address	0 - 63
Register	Low/High Byte
Channel type	Binary/Analogue
Functionality	Input, Output, Input/Output
Analogue measure range	0-10 V, ±10 V, ±20 V, 0-20 mA, 4-20 mA, other
Input delay (ms)	2 - 32
Input cut wire detection	Disabled/Enabled
Output cut wire detection	Disabled/Enabled

Table 4.1 Unit and channel specific parameters.

Hence, the information described above is necessary to configure existing CS31 units, and the least one would expect to find in the database. In order to render the size of the database as small as possible, it was decided to represent the database in a low level format, with each byte containing configuration information. In addition, the database has to be started and terminated by a unique and well recognisable separator. It will also have to contain separators at unit and channel level, to simplify the decoding of the database.

In the definition of the database, the memory is considered divided into segments of 8 bits (1 byte). For simplicity, their values are represented as hexadecimal, i.e. each memory segment ranges between 00 - FF. The earlier mentioned separators are chosen as FF, FE and FD respectively. These values are strictly reserved to partition the database, and must not be used to represent data. In addition to the set of configuration parameters, a couple of variables are introduced; the number of units in the system, and the number of channels of each unit. Figure 4.1 presents a general overview.

1 B - Binary, A - Analogue, I - Input, O - Output.

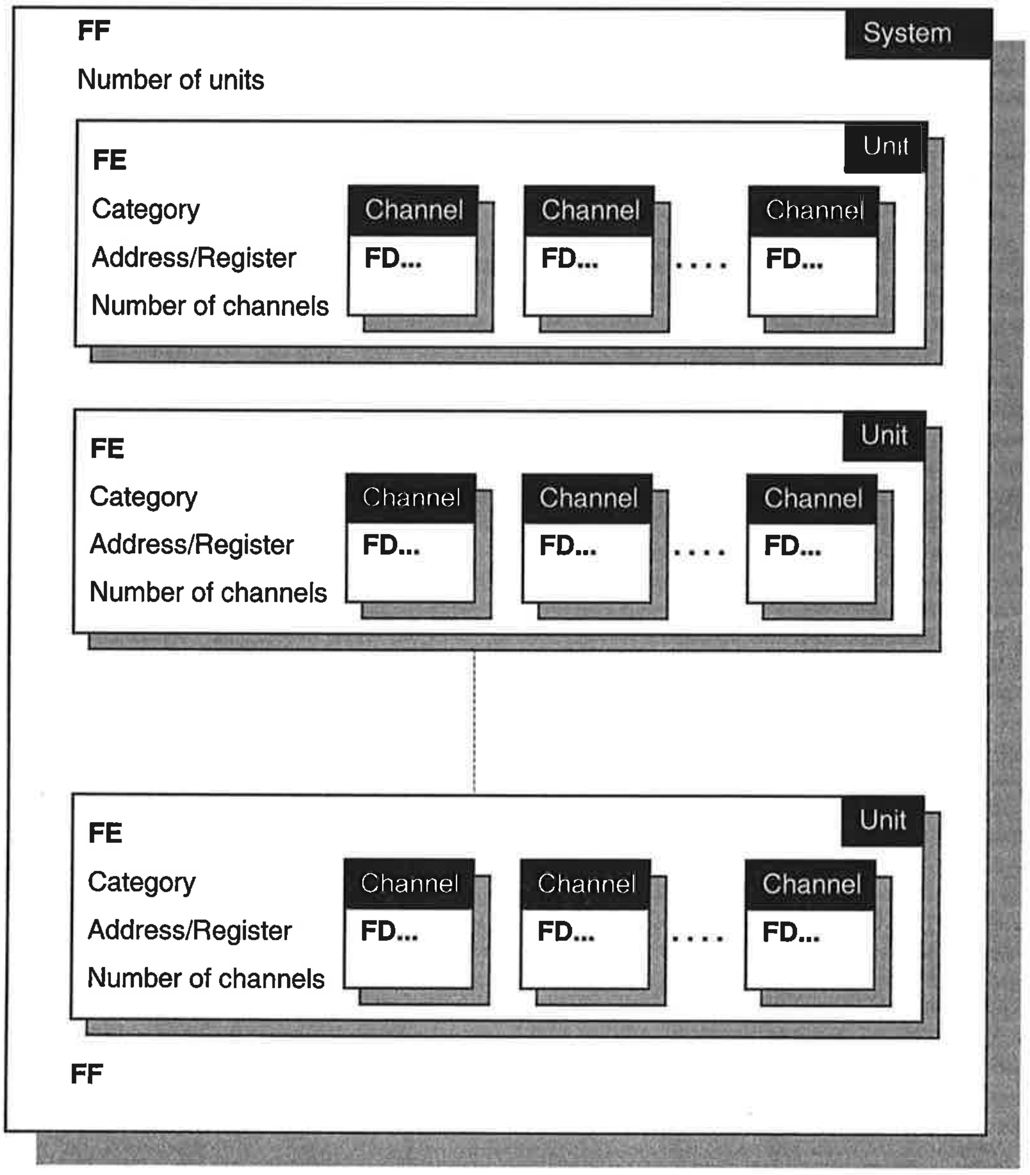


Figure 4.1 General layout of the CS31 database. The contents of the black boxes are to be regarded as comments.

This database contains all relevant configurable information of the CS31. In a worst case today scenario (i.e. 31 I/O units ICFC 16 L1), the database occupies at most 2 Kbytes in the master PLC memory, well within reasonable limits. A complete description of the database is found in appendix A.

4.2 Flexibility

The CS31 is a dynamic, constantly growing automation system. As the market demands improved performance, more powerful PLCs or remote units are released. Hence, the global set of configuration parameters is eligible for extension with new parameters, for example scaling of analogue measure range, but also other, more complex variables. Thanks to an object oriented implementation, CS31CONFIG can be modified to incorporate these variables when the need arises. However, the database represents a more rigid structure, and the definition must therefore encapsulate if not the details (after all, these are not known), at least a general guideline for expansion. Figure 4.2 shows a part of the CS31 database at channel level.

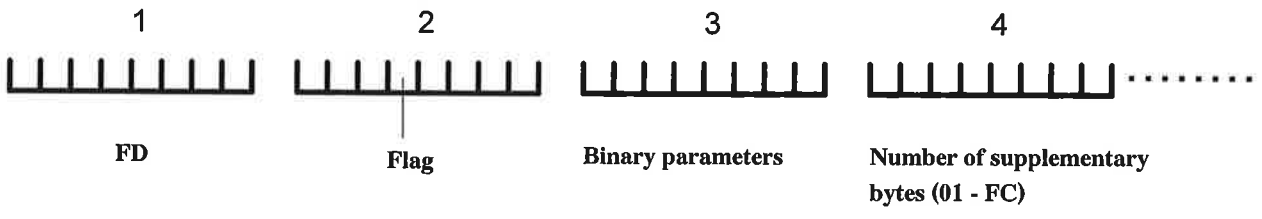


Figure 4.2 The CS31 database at channel level.

Byte 2 contains a flag (bit 5), indicating the presence or not of following bytes containing supplementary information (not used with today's I/O units). As future configuration parameters might be highly complex, byte 4 represents the number of following bytes containing the actual information. Remark that FF, FE, FD are reserved values, so the maximum number of bytes that can be inserted for each channel is 252! The details are implemented in bytes 5 - 256, and will be defined by ABB when the nature of the configuration parameters is better known. Thus, the CS31 database is not limited to the present line of products, but is prepared for future applications.

5. Microsoft Foundation Class Library and Windows Programming

When the general guidelines of the CS31CONFIG was defined, simplicity at the user level was stressed as an important feature. As Microsoft Windows has become a popular and widely used operating system, the choice of developing a Windows application was quite natural, especially considering the well known graphic end user interface standard, stated by the Windows norm. Hence, an operator familiar with other Windows applications, for example Excel or Word, immediately feels comfortable when working in the CS31CONFIG graphic environment, minimising the cost of introductory training. The CS31 is a constantly expanding system that will probably exist for a long time. Hence, its supporting software must be easy to update, and therefore, an object oriented technique was adopted. As a development tool, the Microsoft Visual C++ version 1.51 was chosen. The language C++ is a powerful and widely used programming language suitable for applications of this kind. Also, the main program under development by an external society is written in C++, imposing a significant constraint on the choice.

5.1 Microsoft Foundation Class Library - Overview

The Microsoft Foundation Class Library (MFC) [6], is a group of C++ classes enabling programmers to write applications for the Windows operating system. The class library offers a complete application framework, which defines an architecture for integrating a graphic user interface of an application for Windows, with the rest of the application. The main purpose of the MFC is to reduce the effort required to design and implement applications for Windows. Since the MFC is an object oriented class library, it is easily reused. Instead of editing application framework source code directly, new and specialised classes are derived from those in the library. The derived classes inherit all the behaviour and functionality of their base classes. In addition, programmers often extend their derived classes by adding new member variables and functions, sometimes modifying the existing behaviour by overriding inherited virtual member functions. Figure 5.1 shows a part of the MFC class hierarchy.

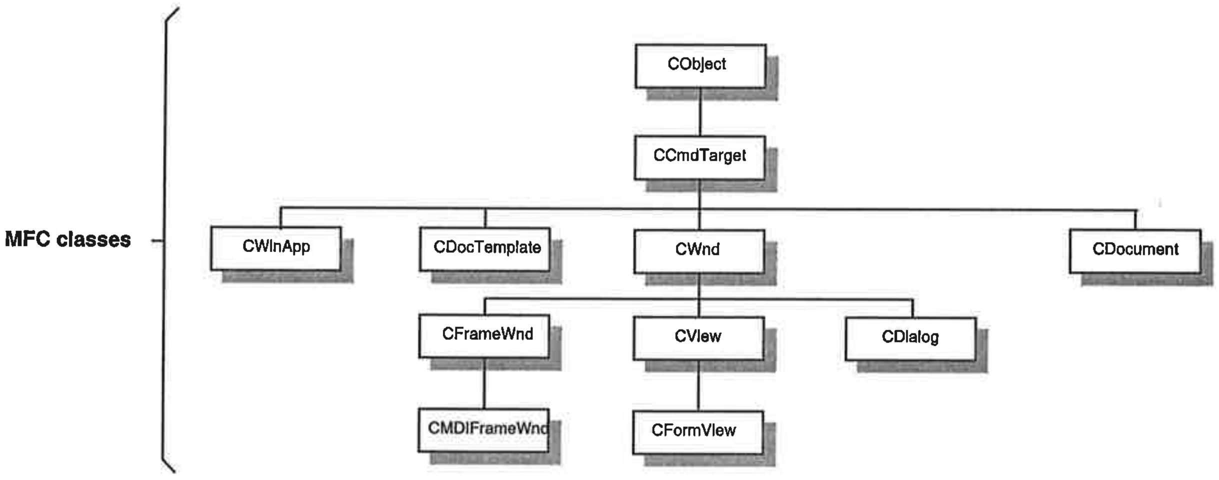


Figure 5.1 Class hierarchy. By convention, class names begin with an uppercase "C".

At run time, a MFC application is a group of co-operating objects, communicating by sending and receiving Windows messages, and by calling each other's public member functions.

5.2 Key Concepts

At the heart of an application for Windows is the application object, typically belonging to a class derived from the base class CWinApp. The application object manages a list of documents and dispatches commands to other objects in the program. Briefly, it co-ordinates the activity of the application. The unit of data that the user works with is the document, derived from class CDocument. The document is responsible for maintaining, loading and storing its own data. The user interacts with a document through a view on the document, derived from CView. A view is visualised by an embedded window in the client area, i.e. the interior of a frame window. It displays the documents data or a part of it, and takes mouse and keyboard input, which it translates into selection and editing actions. Figure 5.2 shows the important relationship between the document and its view. Furthermore, an application can be of type Single Document Interface (SDI), or Multiple Document Interface (MDI). A SDI application allows only one open document at a time, while a MDI application like CS31CONFIG allows several open documents simultaneously. Objects in the user interface, such as menus and buttons, send commands to the documents, views, and other objects in the application. Those objects carry out the commands.

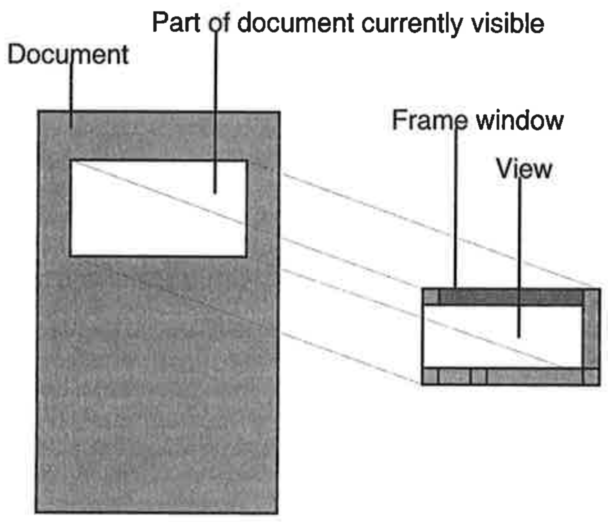


Figure 5.2 Document and view.

Document Templates

To manage the complex process of creating documents with their associated views and frame windows, the application object uses a document template class, CDocTemplate, which is an abstract base class that defines the basic functionality for document templates. One or many document templates are created in the implementation of the application object. A document template defines the relationship between a document class, a view class and a frame window class. An application has one document template for each type of document it supports. CS31CONFIG supports only one type of document and has consequently one document template. Each document template is responsible for creating and managing all the documents of its type.

Communication Flow

The hard core of an application is the application object that manages all other objects belonging to the application. It keeps a list of document templates, used for the creation of a document. The document template is responsible for creating and managing the application's documents. In addition, it links the document with one or several views and frame windows. The document stores the data, and its attached view visualises it. The view is own by a parent window, i.e. a frame window. These object communicate intensively during runtime, and figure 5.3 shows the relationship between them.

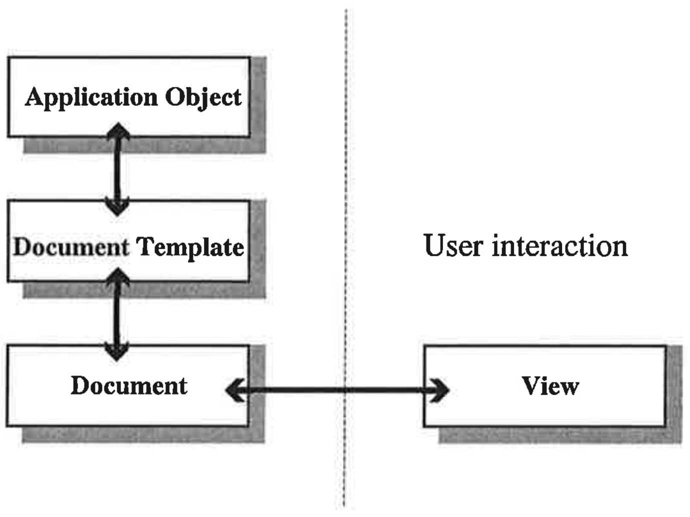


Figure 5.3 The communication flow between different objects.

5.3 Different Windows

Frame Windows

Each application has one and only one main frame window that encapsulates the rest of the application's windows. Usually, it bears the name of the application in its caption (c.f. CS31CONFIG). For a SDI application, there is only one frame window, derived from class CFrameWnd. This window is both the main frame window, and the frame window that encloses the view. For a MDI application, the main frame window is derived from class CMDIFrameWnd, and it allows several documents to be opened simultaneously, and contains consequently several frame windows. As a part of the default implementation of frame windows, a frame window keeps track of a currently active view.

Child Windows

A child window is a window that is confined to the client area of its parent window. For example, the view is encapsulated by its parent, a frame window, which in turn is encapsulated by the main frame. Every child window must have a parent window. The parent window relinquishes a portion of its client area to the child windows, and the child receives all input from this area. The windows class does not have to be the same for each of the child windows of a parent window. This means that an application can fill a parent window with child windows that look differently and carry out different tasks.

5.4 Messages and Message Maps

All applications written for Windows are "message driven". A keystroke, mouse click or other event, cause a Windows message to some part of the application that can respond to the event. There are three kinds of Windows messages: Commands, Standard Windows Messages and Control Notifications.

Commands

A Command is an instruction to an application to perform a certain action. Unlike a function call, a Command is a Windows message that is routed to various command target objects, each of them has an opportunity to carry out the associated instruction. Commands can be sent to frame windows, documents, views, the application itself, and other kinds of objects. These objects are referred to as command targets. Menus, items, buttons, and similar elements of the user interface can cause Windows to generate Commands. These elements are bound to a Command and later recognised by assigning them an ID.

Message Maps

How does a command target know it can handle a Command, and how does any object know it can handle a message? The answer is the message map of the object's class. Each command target object, derived from class CCmdTarget, has a message map. Message maps are tables that connect Commands or other Windows messages, with a certain function, a message handler, that can respond to the Command. Strictly speaking, a message handler is a member function with no argument, returning void. When a command target receives a message, the object's message map is used to determine which handler function to call for the message.

Command Routing

Commands are routed through a standard sequence of command target objects on the assumption that one of them has a handler for the Command. First, the main frame receives the Command. In case of an MDI application, it gives the currently active MDI child window a chance to handle the Command. Because of the standard routing, this frame window now gives its view a chance to handle the Command before checking its own message map. Unlike the frame, the view checks its own message map first and, finding no handler, the view routes the Command to its associated document which checks its message map. If there does not exist a handler in the document, the Command would return to the view and then to the frame window. Finally, the Command would be routed back to the main MDI frame and then to the application object - the ultimate destination of unhandled Commands.

Standard Window Messages and Control Notifications

Standard Window Messages and Control Notifications are sent only to windows and are never routed like Commands from one command target to another. Standard Window Messages are sent to frame windows, dialogue boxes, views and other kinds of windows. Control Notifications are sent from controls, i.e. child windows to their parent windows. For example, when closing an application, a Standard Window Message is sent to the application's main frame window, terminating the application and closing the window. All windows have a message map of its own (as do command target objects).

5.5 Applications under Windows

Instances

As a multitasking system, Windows lets several copies of an application, i.e. instances, run at the same time. In order to keep track of all application objects, Windows assigns a unique ID, an instance handle, to each copy of a running application. Multiple instances of the same application use the same code segment, but each has its own data segment. Windows uses the instance handle to identify the data segment that correspond to a particular instance of an application. When the application begins, Windows passes an instance handle to the corresponding application class.

Creation of an Application

The application object provides overridden member functions for initialisation, execution and termination of the application. Each application using the MFC can only contain one object derived from class CWinApp. This object is already available when Windows calls the WinMain function, a main program co-ordinating the Windows activity, which is done at the initial entry point for a Windows application, for example when a user runs an application. Each time a new instance of an application is demanded, WinMain calls the overridden virtual function CWinApp::InitInstance. The InitInstance creates one or many document templates, depending on the implementation, that in turn create the documents, views and frame windows. After fulfilling the creation phase, Windows calls the OnIdle function, entering the main Windows loop, waiting for messages. To terminate an application, Windows calls the ExitInstance, which terminates the application and reallocates used memory. As with InitInstance, both OnIdle and ExitInstance are virtual member functions of class CWinApp, and can be overridden to perform application specific tasks. For example, when Windows goes on idle, an application can perform application specific background tasks, non crucial to the real time processing.

5.6 Resources

Resources are data objects that are separated from the main body of code in a Windows application. Because of this, resources can be built independently of the rest of the development process. This not only makes a project with a complex user interface easier to manage, but it also renders the application easier to translate into other languages. Resources are often used to describe the contents and the appearance of the user interface objects such as menus and dialogue boxes. A resource file can be created by means of AppStudio, a Visual C++ integrated graphics editor. Another possibility is offered using a common line editor. The term resource template refers to the variables in the resource file determining the state of a resource, for example height and width of a dialogue box, content of a string table or an image of an icon. To most efficiently manage available memory, Windows normally leaves resources on disk until they are needed, for example when the operator invokes a certain dialogue box. Once loaded, a resource is usually placed in the discardable section of the memory, so that memory can be liberated for other tasks if needed. The following predefined resource types are supported by built-in Windows library routines, and are used in CS31CONFIG.

Menus

Menus are used to issue commands.

Icons

Icons are used to represent minimised windows.

Bitmaps

A bitmap is a graphical image stored in a binary format; the basic element of the bitmap is the pixel. In CS31CONFIG, a bitmap is used to define the appearance of the custom command buttons in the toolbar.

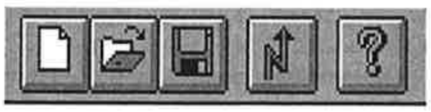


Figure 5.4 The CS31CONFIG toolbar.

Cursor

The cursor shows the position of the mouse or other pointing device on the screen. Programs use different cursors to show a change in the state of current action. For example, when translating the configuration parameters, CS31CONFIG displays an hourglass indicating that the application is busy.

String Tables

A string table is a resource type permitting to store text strings that will be displayed as part of the user interface. Hence, instead of writing the contents of a string in the source code, for example as an argument of the function printf(...) [4], the string is placed in the resource file. When a certain string is needed, it is simply loaded, for instance by means of the member function CString::LoadString(IDS_STRING) [5], which initialises a CString object with the string in the string table identified by IDS_STRING. This simplifies editing the program because all text messages are centralised in the resource file. It is extremely important and time saving if the application is meant for an international market which was clearly a part of the CS31CONFIG specification. By gathering all language dependant messages in the resource file, local ABB societies can easily adapt CS31CONFIG to their native languages.

Dialogue Boxes

A dialogue box is a window, usually a pop-up window (i.e. belonging to the main frame window), or a child window confined to the client area of its parent window (for example the view). Dialogue boxes are used to gather and present user information and represent a two-way communication.

6. Implementation

6.1 General Structure of CS31CONFIG

CS31CONFIG was designed and programmed exploiting the powerful features of the MFC. Hence, it adopts an object oriented approach with a structure and disposition described in the previous chapter. Each CS31 I/O unit is described by its own class, derived from class CDialog. These contains unit and channel specific member variables, assigned to temporarily store the actual settings of the concerned subset of configuration parameters. After the user has entered the actual setting, the dialogue member variables are copied into the documents member variables, where they are stored during execution. The dialogue classes also act as an interface between the user and the document. Configured units are displayed in the view, which contains a list box of units in the network.

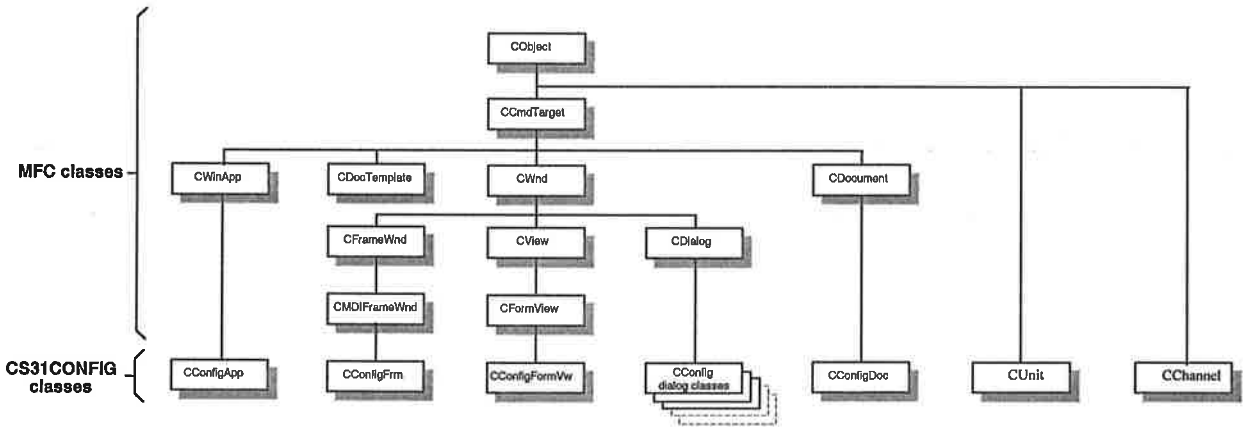


Figure 6.1 Class hierarchy. CS31CONFIG contains 18 dialogue classes.

6.2 The Document Class : CConfigDoc

The document is responsible for storing the actual setting of the configuration parameters. In MFC, documents are based on the class CDocument. Thus, the CS31CONFIG document class, labelled CConfigDoc, is a subclass derived from CDocument. It contains one member variable of class CObList, a collection template class provided by the MFC. An object of class CObList can be considered as a doubly linked list, where each element is an object of a class derived from the base class CObject. The CObList of CS31CONFIG contains up to 31 objects of class CUnit (CS31CONFIG specific), derived from CObject. CUnit is the class describing unit specific information, and contains in turn embedded objects of class CChannel, also derived from CObject, describing channel specific information.

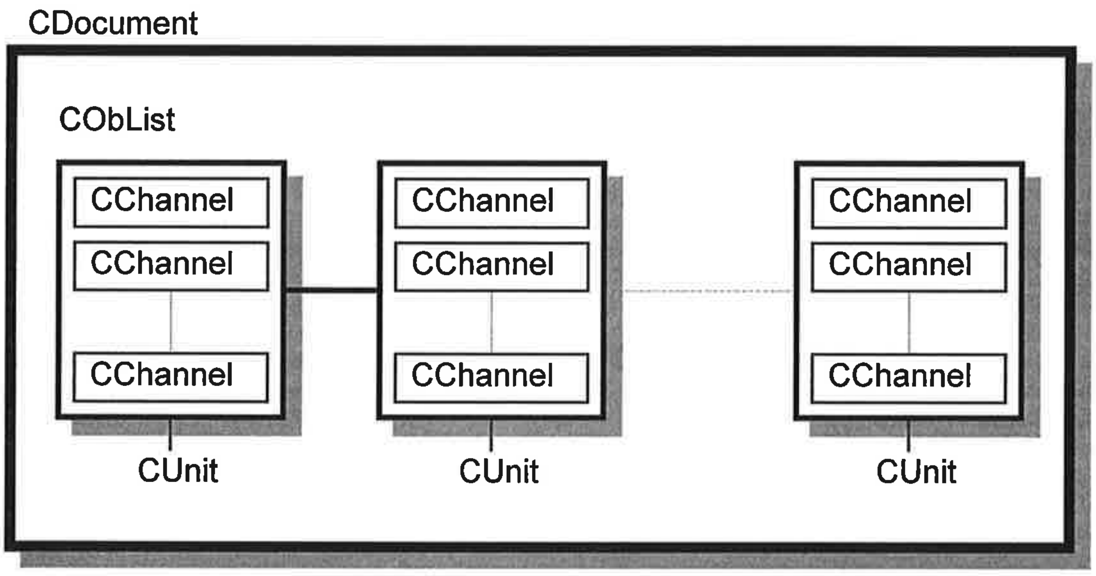


Figure 6.2 The document contains an object of class CObList.

All unit and channel specific configurable parameters, i.e. the document's data, correspond to protected member variables. In other terms, they are inaccessible outside the class to which they belong (CUnit and CChannel respectively). The main reason is to assure that "...a minor change of the problem should only impose changes in one module, or at most, a limited number of modules..." as stated by [3]. To access member variables of class CUnit/CChannel, or rather copies of them, one must though rely on the public member functions declared in tables 6.1 and 6.2.

```
class CUnit : public CObject {
//Attributes
protected:
        int m_address;
        BOOL m_highbyte;
        int m_category;
        int m_NoOfChannels;
        CChannel m_channel[31];
        CString m_UserComment;
//Operations
public:
        void AddUnitData(...);
        void GetUnitData(...);
        CString TranslateUnit2DBase();
        CString TranslateUnit2MAIL(); }
```

Table 6.1 Declaration of class CUnit. By convention, member variable names begin with a lowercase "m_".

```
class CChannel : public CObject  {
//Attributes
protected:
        int m_config:  //[i,o,io]
        BOOL m_binary;
        int m_range;
        int m_delay;
        BOOL m_idet, m_odet;
        int m_NoOfSuppl;
//Operations
public:
        void AddChannelData(...);
        void GetChannelData(...);
        CString TranslateChannel2DBase();
        CString TranslateChannel2MAIL();  }
```

Table 6.2 Declaration of class CChannel.

6.3 The View Class : CConfigFormVw

The document object defines, stores, and manages the application's data. However, all user interaction with the document is handled through a view object. In MFC, a view is an object derived from CView (or from another CView derived class such as CFormView). The view displays the document, and acts as an intermediary between the user and the document for input, selection and editing in the document. A given view is always associated with only one document. Each CS31CONFIG document uses only one view class. On the contrary, some applications can use several view classes to visualise the documents data in different ways. The views, each framed and owned by its own frame window, occupy the client area in the application's main frame window. These frame windows are children of the main frame.

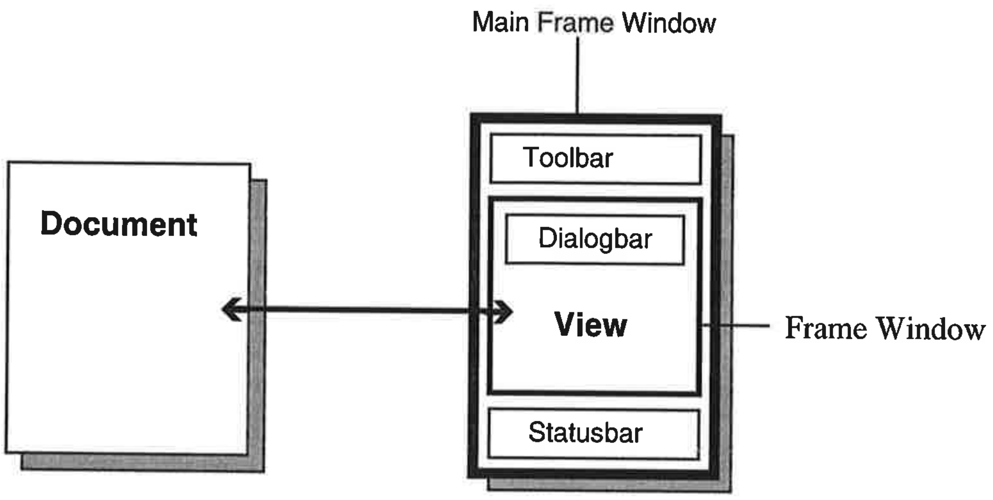
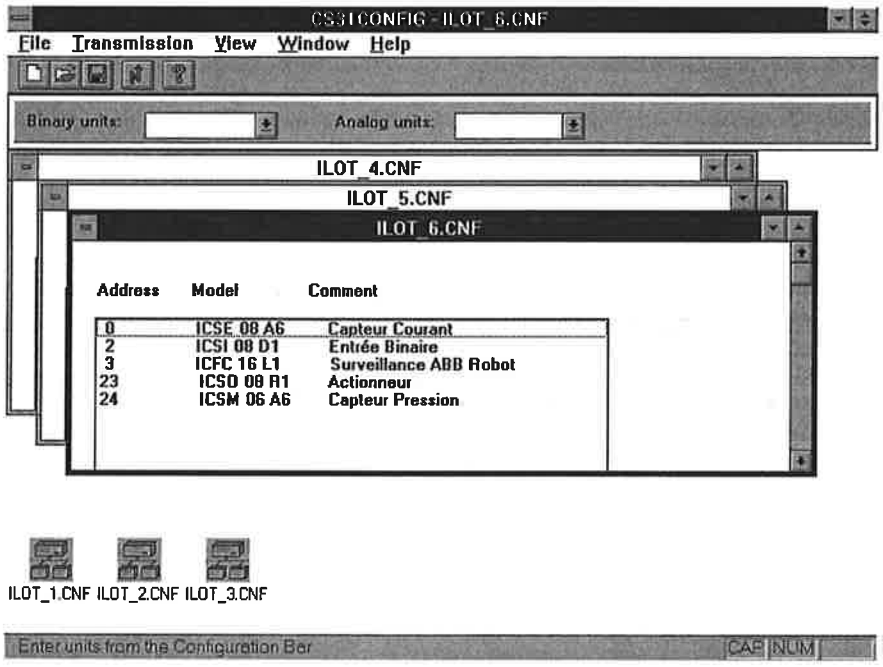


Figure 6.3 Exchange of data between the document and its view.

If the view contains controls (such as list boxes, buttons etc.) which is the case of CS31CONFIG, the application's view class has to be derived from CFormView instead of CView. CConfigFormVw keeps a list box with configured units in the network. The window representing the view can be iconised to provide an easier overview when working with multiple documents.



Address	Model	Comment
0	ICSE 08 A6	Capteur Courant
2	ICSI 08 D1	Entrée Binaire
3	ICFC 16 L1	Surveillance ABB Robot
23	ICSO 08 R1	Actionneur
24	ICSM 06 A6	Capteur Pression

Enter units from the Configuration Bar

Figure 6.4 CS31CONFIG is a MDI application and can manage several documents each with an attached view simultaneously.

In order to create objects describing I/O units, CConfigFormVw uses an embedded object of class CDialogBar, which encapsulates two combo boxes, cf. figure 6.5. By selecting a CS31 I/O unit in one of the combo boxes, CS31CONFIG invokes a corresponding dialogue box, reflecting the choice. Upon user confirmation, the unit is visualised in the list box described above, which represents the system. Hence, the view renders a part of the document visible namely address, type of unit, as well as a user comment. All other information is filtered to simplify the system overview. By simply selecting desired units in the view, CS31CONFIG renders all parameters regarding the selected unit visible. This is handled via an object derived from class CDialog.

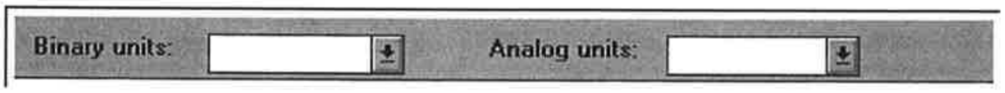
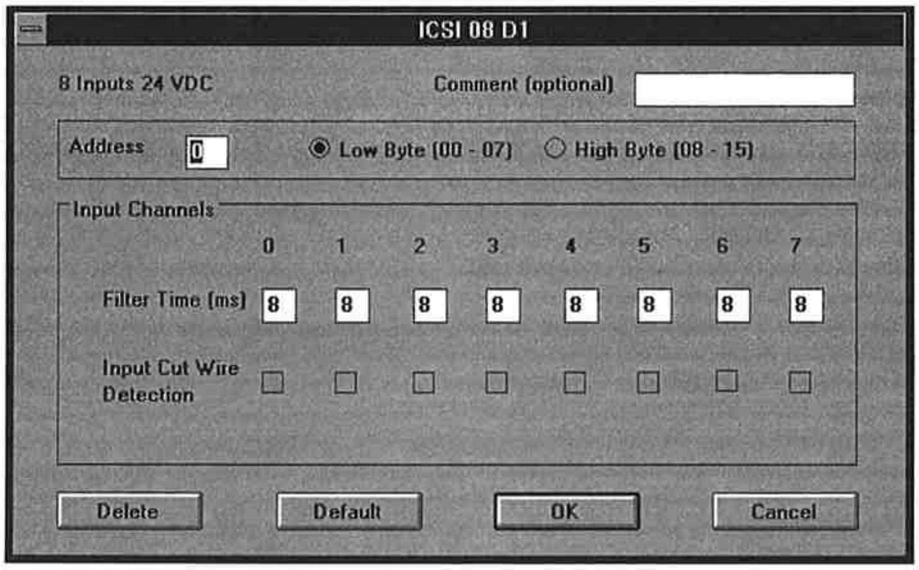


Figure 6.5 Dialogue bar with embedded combo boxes.

6.4 CS31CONFIG Dialogue Classes

CS31CONFIG uses CDialog derived classes to describe the CS31 I/O units. Each unit is assigned its own class, with its own set of member variables corresponding to the hardware. To render the configurable parameters visible, each unit is also associated with a dialogue resource template. These were created using the graphical editor AppStudio, simplifying the design of the user interface. By selecting I/O units from the dialogue bar in the view, CS31CONFIG creates objects of the associated class, and invokes the corresponding dialogue box. Thanks to the simplicity of the graphical user interface, the user can rapidly enter the actual settings. Upon confirming by clicking «OK», the configuration parameters are passed to the document, where they are stored. Thereafter, the view is updated to reflect the changes, i.e. insertion of an element in the list box. Once a unit has been configured, it can be retrieved by selection from the same list box in the view. Upon selection, the corresponding unit is traced in the documents object of class CObList. The stored parameters are used to reinitialise the member variables of the object of the dialogue class, which is created. Thereafter, the dialogue box is invoked as in the initial configuration phase, and the user can execute changes. Serious effort is spent to render the configuration process fail-safe. Therefore, each user provided dialogue member variable is validated before storing. Other routines prevent the user from configuring units at the same address, as well as exceeding the limit 31 units on the CS31 system bus.


ICSI 08 D1

8 Inputs 24 VDC

Comment (optional):

Address: 0 [✓ Low Byte (00 - 07)] High Byte (08 - 15)

Input Channels

	0	1	2	3	4	5	6	7
Filter Time (ms)	8	8	8	8	8	8	8	8
Input Cut Wire Detection	☐	☐	☐	☐	☐	☐	☐	☐

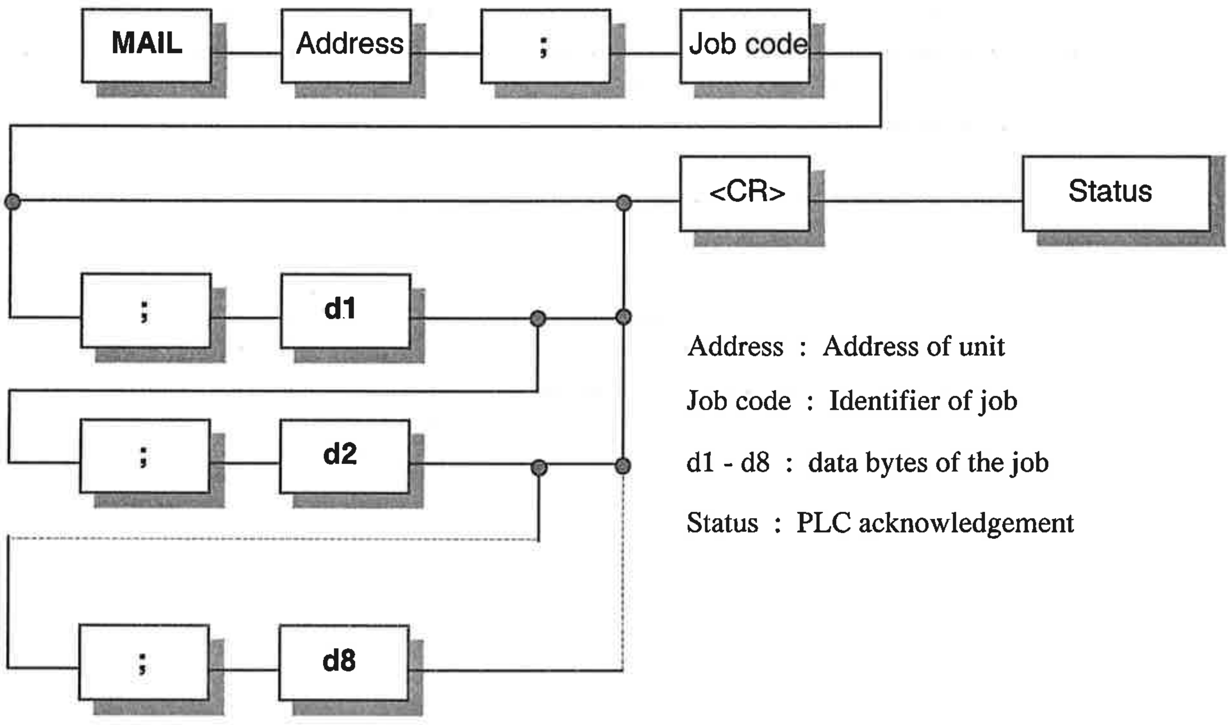
Figure 6.6 A dialogue box presents information and allows the user to rapidly enter configuration parameters.

6.5 Transmission

Data processing and transmission are handled by a message handler of class CConfigDoc. Until now, the user provided settings of the configuration parameters are stored in the CUnit and CChannel member variables, as described in the beginning of this chapter. These have to be processed and translated into either MAIL commands, or the CS31 database. This choice is provided by the user when entering the actual target PLC. The transmission message handler then sets up a communication link by means of library routines.

MAIL

Currently existing CS31 PLCs are prepared to receive ASCII characters via a serial port communication. The syntax of the command MAIL is defined by figure 6.7



Address	:	Address of unit
Job code	:	Identifier of job
d1 - d8	:	data bytes of the job
Status	:	PLC acknowledgement

Figure 6.7 The CS31 command MAIL

CS31CONFIG uses one unit and six channel related MAILs, identified by their unique job code cf. table 6.3, to configure the CS31 system. Initially, a unit specific MAIL affecting all channels is always transmitted, resetting all channel specific configuration parameters to their default values. Thereafter, each particular channel is adjusted to the desired setting by means of channel specific MAILs.

Command	Job code	Level
Resetting configuration parameters to default setting	233	Unit
Activating cut wire detection of an input	224	Channel
Activating cut wire detection of an output	225	Channel
Configuration of a channel as an input	162	Channel
Setting the input delay of a channel	166	Channel
Setting analogue input measure range	170	Channel
Setting analogue output measure range	171	Channel

Table 6.3 MAILs used by CS31CONFIG

Table 6.3 shows the seven MAILs representing a complete base for configuring any CS31 system, i.e. all possible combinations of parameters can be attained. To assure correct transmission, each character received by a CS31 PLC is echoed, i.e. retransmitted to the serial port, where its read by CS31CONFIG. If the retransmitted character differs from the character sent, transmission is interrupted, and CS31CONFIG notifies the operator. In addition, after receiving a complete MAIL, the PLC sends a status response, indicating if the command could be executed or not, and in the latter case the source of potential errors. This tightly links the configuration procedure with another planned tool in the CS31 programming software; the diagnostic module. In addition to the seven MAIL commands listed above, the command MAIL also encapsulates some of the commands for diagnostic functions, such as interrogation of configuration parameters, and acknowledging errors on remote modules. This implies an intensive information flow between the configuration/diagnostic modules, since they partly describe different angles of the same process. The ambition of co-operating modules falls outside the scope of this master thesis, and will have to be defined and implemented by ABB and the external software developer. However, CS31CONFIG is prepared to import and export information, and can be modified to support interaction with a diagnostic application.

CS31 Database

Since the database does not represent an explicit command, but purely data, the actual process of configuration is delegated to the PLC itself. The role of the CS31CONFIG is rather to act as a backup tool, launching a configuration process, supplying the PLC with the actual settings of the configuration parameters. Thus, this renders the configuration process hardware based, as opposed to today's more software oriented version, which demands profound knowledge of the PC33 or the MAIL syntax. A desired effect is to introduce more self supported, intelligent systems. From the software point of view, this simplifies the transmission phase, keeping in mind that the PLC will be responsible for the execution and verification of the command, transforming the role of CS31CONFIG into an intelligent graphic user interface.

The value of one byte in the database is transmitted as two ASCII characters, representing the hexadecimal value of the byte, i.e. ranging between 00 - FF. The future PLC will compress these two characters into one byte, reducing the size of the database by two. As with the MAIL command, the CS31CONFIG correlates the echoed ASCII characters (feature implemented on future CS31 PLCs), to assure a correct transmission. In order to fully exploit the hardware oriented configuration process, diagnostic interaction should follow the same concept, i.e. the responsibility is delegated to the PLC. The diagnostic tool is in parallel with CS31CONFIG, a co-ordinator of the activity, and hopefully, the two modules running in close co-operation, will fulfil an important part of a new generation PLCs, more sensitive and better adapted to real world situations.

7. Summary

This master thesis has modelled and implemented a configuration tool, CS31CONFIG, for the ABB Procontic CS31 automation system. The configuration process serves to set the unit and channel associated configuration parameters at appropriate values. Depending on the actual PLC, CS31CONFIG will either use the existing command MAIL, or transmit a database containing all configuration information, to a CS31 PLC. Hence, CS31CONFIG is both compatible with existing PLCs, as well as prepared for a future generation of PLCs. The structure of the database was defined during this master thesis.

In the definition of the database, there were several constraints to consider, especially concerning the compactness, and the simplicity of the decoding. In addition, the database had to adopt an expandable structure to stay compatible with planned, future configuration parameters of the CS31. Naturally, the complexity of the problem increased as the constraints grew more significant, particularly since the specification concerning future features were quite vague.

The fact that the CS31 is an expanding system imposed a major concern regarding the implementation of the CS31CONFIG. Since CS31 is a product that will exist for a long time, its supporting software must be easy to update. This was solved by adopting an object oriented technique, and the programming was carried out in C++.

The part of the specification concerning the simplicity of the user interface was easily solved exploiting the powerful MFC, which in parallel with the graphic editor AppStudio, made the design process quite fast. The result is very logic and user friendly, rendering introductory training almost unnecessary.

As an independent tool, the CS31CONFIG fulfils its part of the specification, and runs as expected. However, regarding the co-operation with a diagnostic tool, a part of the initial specification that could not be attained, as well as the integration with the main program, there remain quite a few things to define and to implement. It concerns especially the information flow between the two tools, and the main program, which appears to be rather complex. Also, ABB must more clearly define the specification of the diagnostic tool, in order to understand what impact it will have on the CS31CONFIG.

A. Significance of Bits in the CS31 Database

Level	Byte	Bit(s)	Value*	Configuration Parameter
System	1	1 - 8	[FF]	Main separator
	2	1 - 8	[00 - 1F : 0 - 31]	Number of units
Unit	1	1 - 8	[FE]	Unit separator
	2	1 - 3	[00 : Binary Input] [01 : Analogue Input] [02 : Binary Output] [03 : Analogue Output] [04 : Binary Input/Output] [05 : Analogue Input/Output]	Unit functionality
		4 - 8	Not used	
	3	1 - 6	[00 - 3F : 0 - 63]	Address
		7	[00 : Low Byte] [40 : High Byte]	Register
		8	Reserved	
	4	1 - 8	[00 - FC : 0 - 252]	Number of channels

Level	Byte	Bit(s)	Value*	Configuration Parameter
Channel	1	1 - 8	[FD]	Channel separator
	2	1	[00 : Analogue channel]	Channel type
			[01 : Binary channel]	
		2	[00 : Input channel off]	Functionality
			[02 : Input channel on]	
		3	[00 : Output channel off]	Functionality
			[04 : Output channel on]	
		4	[00 : Byte 3 off]	
			[08 : Byte 3 on]	
		5	[00 : Byte 4 off]	
			[10 : Byte 4 on]	
		6 - 8	[00 : Binary channel]	Analogue measure range
			[20 : 0 - 10 V]	
			[40 : ± 10 V]	
			[60 : 0 - 20 mA]	
			[80 : ± 20 mA]	
			[A0 : 4 - 20 mA]	
			[C0 : Other range]	
	3	1 - 6	[00 - 3E : 0 - 62]	Input delay
		7	[00 : Disabled]	Input cut wire detection
			[40 : Enabled]	
		8	[00 : Disabled]	Output cut wire detection
			[80 : Enabled]	
	4	1 - 8	[01 - FC : 1 - 252]	Number of supplementary bytes
	5 - 256	1 - 8	[00 - FC]	Supplementary information (format not defined)

***Syntax: [Identifier : Interpretation].** Identifier is the hexadecimal value to superposition a byte whilst interpretation is the corresponding physical value. When interpretation is omitted, the parameter does not have a physical correspondence. Remark: a byte is considered to be oriented as [87654321].

References

[1] ABB Control, ABB Procontic CS31, Edition 02, 1995.

[2] ABB Schalt- und Steuerungstechnik GmbH, 907 PC 33. Programming and Test Software. Operating Manual GATS 1339 31 R2001, Edition 08, 1992.

[3] Holm, Per, Objektorienterad programmering och Simula, 2nd Edition, KF-SIGMA, 1992.

[4] Kernigham, B.W., & Ritchie, D.M., Le langage C, Prentice Hall, 1992.

[5] Lebatard, Alain, Le grand livre de Visual C++ 1.5, Editions Micro Application, 1994.

[6] Microsoft Press, Microsoft Visual C++ version 1.5 documentation, Microsoft Corporation, 1992.