**NTNU**
**Norwegian University of Science and Technology**

**Faculty of Physics, Informatics and Mathematics**

# *DIPLOMA THESIS*

## *FOR*

## *STUD.TECHN. TROND LIENG*

## *FACULTY OF PHYSICS, INFORMATICS AND MATHEMATICS*

## *NTNU*



*Date due:* *May 5, 2000*

***DISCIPILINE:*** ***Applied Physics***

***Title:*** ***«Implementation of Profibus in the ISOLDE control system»***

*Purpose of the work:*
*Develop Visual Basic functions for controlling a Profibus network. Implement Profibus in the control system of ISOLDE by developing complete Equipment Modules using Profibus and Visual Basic, including interfaces (control panels). Evaluate its performance and see how equipment from different manufacturers works together with each other and with the control system. Provide some documentation for how to create an Equipment Module.*

*This work is to be carried out at CERN, Geneva, under the supervision of Ove Jonsson.*



*Trondheim, November 5 , 1999*

| | | |
|---|---|---|
| *Steinar Raaen* | Ove C. Jonsson | *Kåre Olaussen* |
| *Head of Department* | *Supervisor at CERN* | *Professor* |
| *Department of Physics* | | *Department of Physics* |

# Preface

This thesis is the concluding part of my education at the Norwegian University of Science and Technology (NTNU). It leads to the title Sivilingeniør (B.Sc./M.Sc.). The work has been done at CERN in Geneva, in the ISOLDE group.

The work has been interesting and given insight into technical and physical areas previously unknown to me. It has also given me the opportunity to work in an international environment at CERN and thereby meet a lot of interesting people.

I would like to thank some of the persons that have helped me during this work. My supervisor at CERN, Ove Jonsson, deserves special thanks for giving me a position as Technical Student at CERN and for showing me the way in this work. Professor Kåre Olaussen at the Institute of Physics at NTNU has helped with the administrative and given me valuable feedback on this thesis. Joakim Cederkall in ISOLDE and Frank Locci in the PS Control Group have helped me when I have met obstacles, and should be thanked as well.

It is my hope that this thesis will be useful for someone in the future, even though the replacement of the ISOLDE control system is currently being discussed.

Geneva 05.05.2000

# Table of Contents


**ABSTRACT** ..... **5**

**1 INTRODUCTION** ..... **6**

1.1 CERN AND ITS HISTORY ..... 6
1.2 CERN'S ACCELERATORS ..... 7
*1.2.1 The Proton Synchrotron (PS)* ..... *8*
1.3 ISOLDE ..... 9
*1.3.1 REX-ISOLDE* ..... *10*
1.4 AIM OF THIS THESIS ..... 11

**2 THE TECHNICAL BACKGROUND OF THIS WORK** ..... **13**

2.1 SOME ADVANCED VISUAL BASIC PROGRAMMING ..... 13
*2.1.1 Dynamic Link Library (.DLL) files* ..... *13*
*2.1.2 ActiveX controls* ..... *14*
*2.1.3 Timing* ..... *15*
2.2 THE ISOLDE CONTROL SYSTEM ..... 15
*2.2.1 Hardware architecture* ..... *16*
*2.2.2 The software architecture* ..... *17*
2.3 PROFIBUS ..... 21
*2.3.1 Introduction* ..... *21*
*2.3.2 The ISO/OSI reference model and the technical specifications of Profibus* ..... *22*
*2.3.3 DP-Profibus* ..... *23*
*2.3.4 Installing and operating a DP-Profibus network* ..... *24*
*2.3.5 The Profibus set-ups used in this work* ..... *25*
2.4 DIGITAL/ANALOGUE AND ANALOGUE/DIGITAL CONVERTERS ..... 26
*2.4.1 DACs* ..... *27*
*2.4.2 ADCs* ..... *27*
2.5 QUADRUPOLES ..... 29
*2.5.1 Focusing quadrupoles* ..... *29*
*2.5.2 Steering quadrupoles* ..... *30*
*2.5.3 The REX quadrupoles* ..... *30*

**3 CONTROLLING PROFIBUS WITH VISUAL BASIC** ..... **31**

3.1 SIMATIC NET DP-5412 ..... 31
*3.1.1 The declarations module (Dplib)* ..... *31*
*3.1.2 The procedures module (ProfiProcedures)* ..... *35*
3.2 APPLICOM® PROFIBUS ..... 39
*3.2.1 The declarations module (Applicom)* ..... *39*
*3.2.2 The procedures module (ProfiProcedures)* ..... *41*

**4 THE EQUIPMENT MODULES** ..... **45**

4.1 THE FEC ..... 45
4.2 THE DATABASE ENTRIES ..... 45
*4.2.1 The FEC* ..... *46*
*4.2.2 The Class(es)* ..... *46*
*4.2.3 The Equipment* ..... *47*
4.3 A GENERAL EQUIPMENT MODULE ..... 47

*4.3.1 The User Control* ..... 48
4.4 THE QP_NT EQUIPMENT MODULE ..... 50
*4.4.1 Hardware set-up* ..... 50
*4.4.2 Database Entries* ..... 50
*4.4.3 The ActiveX Control code* ..... 51
*4.4.4 The Control Panel* ..... 55
4.5 THE QS_REX EQUIPMENT MODULE ..... 58
*4.5.1 Hardware set-up* ..... 58
*4.5.2 Database Entries* ..... 58
*4.5.3 The ActiveX Control code* ..... 60
*4.5.4 The Control Panel* ..... 64

**5 DISCUSSION ..... 65**

5.1 PROGRAMMING FOR THE ISOLDE CONTROL SYSTEM ..... 65
*5.1.1 Programming environment* ..... 65
*5.1.2 Access to the control system* ..... 65
*5.1.3 General status of the control system and its documentation* ..... 66
5.2 EVALUATION OF THE PROFIBUS HARDWARE ..... 66
*5.2.1 The Communication Processors* ..... 67
*5.2.2 The slave modules* ..... 67
5.3 PERFORMANCE OF DP-PROFIBUS IN THE ISOLDE CONTROL SYSTEM ..... 69
*5.3.1 Adaptability* ..... 69
*5.3.2 Performance* ..... 70
*5.3.3 Full-scale installation* ..... 72
5.4 THE EQUIPMENT MODULES ..... 73
5.5 THE USER INTERFACES (CONTROL PANELS) ..... 73

**6 CONCLUSIONS ..... 75**

**APPENDIX A INSTALLING PROFIBUS ON A PC ..... 76**

A.1 SIMATIC NET DP-5412 ..... 76
*A.1.1 Hardware Installation* ..... 76
*A.1.2 Software and License Installation* ..... 76
*A.1.3 Setup and Configuration of the Card and the Profibus* ..... 76
*A.1.4 Using the test program DPN Demo* ..... 79
*A.1.5 Creating an equipment module* ..... 79
A.2 APPLICOM ..... 79
*A.2.1 Hardware Installation* ..... 79
*A.2.2 Software Installation* ..... 80
*A.2.3 Configuring the Bus* ..... 80

**APPENDIX B CODE LISTING OF OTHER MODULES ..... 82**

B.1 THE QP_NTAPPLICOM USER CONTROL ..... 82
B.2 THE QS_NT USER CONTROL ..... 83
B.3 THE QP_REX USER CONTROL ..... 86
B.4 THE CONTROL PANEL FOR THE QS_NT EQUIPMENT MODULE ..... 87
B.5 THE CONTROL PANEL FOR THE QP_REX EQUIPMENT MODULE ..... 89
B.6 THE CONTROL PANEL FOR THE QS_REX EQUIPMENT MODULE ..... 91

**REFERENCES ..... 93**

# Abstract

At CERN's facility for producing short lived radioactive isotopes, ISOLDE, parts of the control system are getting old and outdated. To ensure safe and stable operation of the facility as well as easier maintenance, more modern standards are being gradually installed.

In the future, the widely used industrial fieldbus called Profibus will replace a several of the existing interfaces between the equipment in the facility and the computers of the control system. This work has consisted of implementing Profibus to the control system and testing its performance. The implementation has for a great part consisted of writing so-called ActiveX Controls in the Visual Basic programming language. The testing has included the study of Profibus communication processors and input/output modules from two different vendors to see how these interact with each other and with the ISOLDE control system.

The work has shown that Profibus is well suited for its planned use at ISOLDE, and that all the Profibus equipment that has been tested work as intended. However, for reasons of simplicity and economy, it is recommended that communication processors from applicom international and input/output modules from Wago be used wherever possible. The more versatile input/output modules from Simatic can be installed where special concerns make it necessary.

This thesis includes detailed descriptions and examples of how to make the ActiveX Controls called Equipment Modules for the ISOLDE control system. This part of the ISOLDE documentation does not presently exist elsewhere.

# 1 Introduction

This chapter introduces the work behind this thesis and the institutions in which it has taken place.

First, CERN and ISOLDE will be presented as a general background of where this work has been done, and to put it in a larger context.

At the end of this chapter, an overview of the rest of the contents of this paper will be given, as well as the aim of the work that has been done.

## 1.1 CERN and its history

CERN or "European Organization for Nuclear Research" as the official name now is*, is the world's largest particle physics research centre [1]. It was established in 1954 as a joint venture between 12 European countries, one of the first after World War II. It was thereby one of the first attempts to rebuild the relationships between former enemies. This policy later led to the foundation of several pan-European organisations, including the EU.

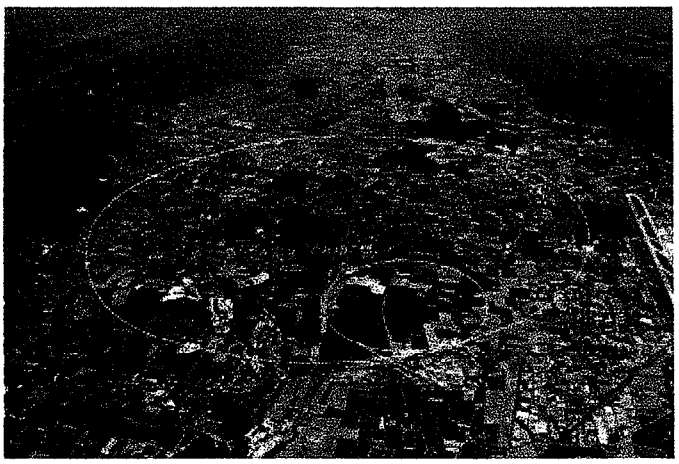


**Figure 1-1: The footprint of the LEP (big circle) and the SPS (smaller circle) accelerators 1. CERN's main site is located where the two circles join. Geneva airport is to the right. (Photo: CERN Photo)**

Since the beginning, the number of member states has grown to 20. In addition to the member states, there are 7 observer states for the time being†.

The main site of CERN is located just outside Geneva on the border between Switzerland

* The acronym CERN comes from the name of the council that preceded the organisation, which was "Conseil Européen pour la Recherché Nucléaire". Today, CERN is perhaps best known as "European Laboratory for Particle Physics", although this is not the official name.)

† These are the member states (year of accession in parathesis): Austria (1959), Germany (1953), Portugal (1986), Belgium (1953), Greece (1953), Slovak Republic (1993), Bulgaria (1999), Hungary (1992), Spain (1961), Czech Republic (1993), Italy (1953), Sweden (1953), Denmark (1953), Netherlands (1953), Switzerland (1953), Finland (1991), Norway (1953), United Kingdom (1953), France (1953), Poland (1991). States with Observer status are Israel, Japan, the Russian Federation, Turkey, the United States of America, the European Commission and Unesco.

and France. The purpose of the site itself is to provide physicists with accelerators that meet research demands at the limits of human knowledge. About 3000 persons are employed at CERN. In addition there are about 6500 scientists, among them half of the world's particle physicists, who use CERN's facilities. They represent 500 universities and over 80 nationalities.

The aim of CERN is to study nature and structure of matter through particle physics. This is the study of the fundamental constituents of matter and their interactions. However, which particles are regarded as fundamental has changed with time as physicists' knowledge as improved, partially thanks to CERN.

Europe had dominated progress in particle physics before World War II, from the discovery of the electron to the atomic nucleus and its constituents, from special relativity to quantum mechanics. Sadly, the conflicts of the 1930s and 40s interrupted this, as many scientists had to leave for calmer shores. After the end of the conflicts, new intensive research could start. By the early 50s, the Americans had understood that further progress needed more sophisticated instruments, and that investment in basic science could drive economic and technological development. While scientists in Europe still relied on simple equipment based on radioactivity and cosmic rays, powerful accelerators were being built in the United States. Tabletop experiments were being overtaken by projects involving large teams of scientists and engineers. It was obvious that no European country had the possibility to make similar efforts alone, so CERN was created as a co-operation to obtain front-line research in Europe.

## *1.2 CERN's accelerators*

CERN's accelerator complex is the most versatile in the world and represents a considerable investment. It includes particle accelerators and colliders that can handle beams of electrons, positrons, protons, antiprotons, and heavy ions such as oxygen, sulphur, and lead. Each type of particle is produced in a different way, but then passes through a similar succession of acceleration stages, moving from one machine to another. The first steps are usually provided by linear accelerators, followed by the larger circular machines. CERN has 10 accelerators altogether, the biggest being the Large Electron Positron collider (LEP) and the Super Proton Synchrotron (SPS).

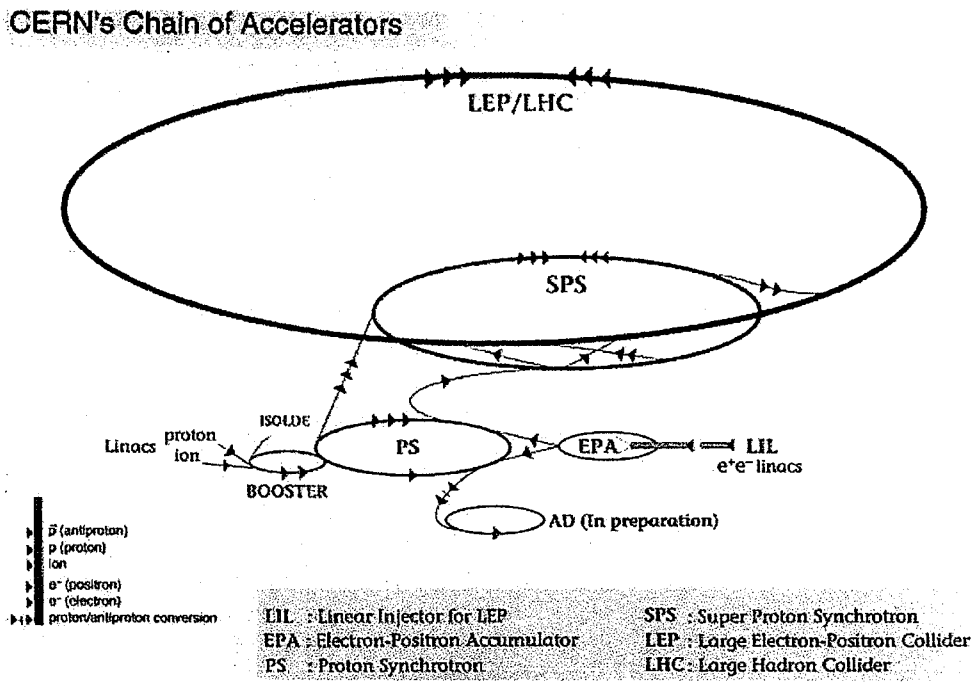



**Figure 1-2: The CERN accelerator complex [1].**

CERN's first operating accelerator, the Synchro-Cyclotron, was built in 1954, in parallel with the Proton Synchrotron (PS). The 1970s saw the construction of he SPS, at which Nobel-prize winning work was done in the 1980s. The SPS continues to provide beams for experiments and is the final link in the chain of accelerators providing beams for the 27 kilometer LEP machine. The LEP will be dismantled during the autumn of 2000 to make space for CERN's next big machine, due to start operating in 2005: The Large Hadron Collider (LHC).

### 1.2.1 The Proton Synchrotron (PS)

The PS celebrated its 40 years anniversary in 1999, and is today the backbone of CERN's particle beam factory, feeding other accelerators with different types of particles.

The PS complex can accelerate all stable and electrically charged particles (electrons, protons), their antiparticles (positrons, antiprotons), and different kinds of heavy ions (oxygen, sulfur and lead).

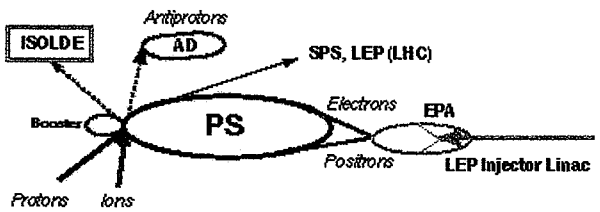



**Figure 1-3: The PS complex with the PS-Booster and ISOLDE [1].**

These particles are produced in ion sources, for instance are protons obtained from hydrogen gas using a "duoplasmatron".

The first stage of acceleration takes place in a linear accelerator (Linac), and each type of particle has its own. The final energies are 500 MeV for electrons, 4.2 MeV/nucleon for lead (Pb) ions, and 50 MeV for protons. For proton and heavy ion beams then follows a "Booster" synchrotron to increase the energy up to 1.4 GeV.

In addition to being sent on to other accelerators, proton beams from the PS-Booster are also used for physics experiments, e.g. ISOLDE.

## 1.3 ISOLDE

ISOLDE (Isotope Separator On-Line) is a facility dedicated to the production of a large variety of radioactive ion beams for a great number of different experiments [2,3]. The research areas range from nuclear and atomic physics, solid-state physics and material science to life sciences through nuclear medicine (see Figure 1-4 below).

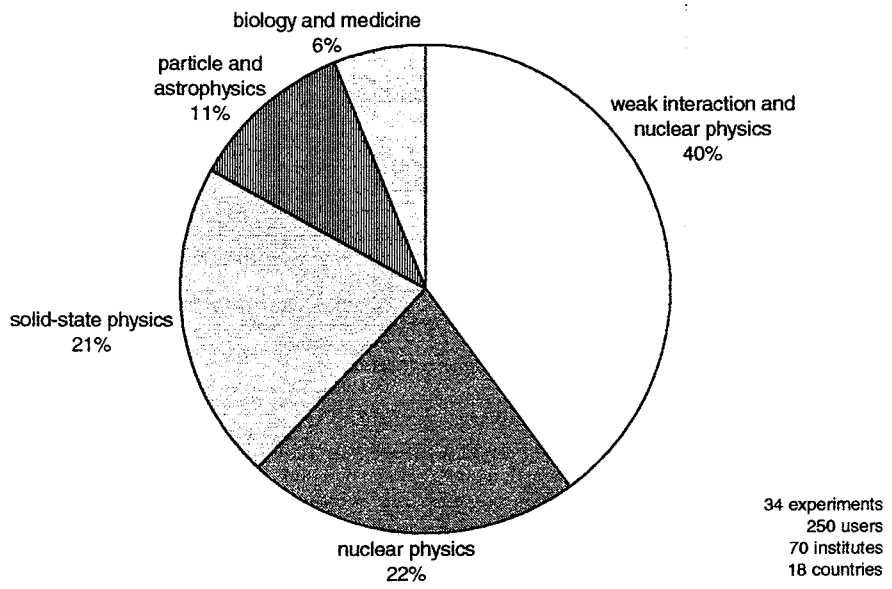



**Figure 1-4: Some key figures about the range of experimental subjects and users of the ISOLDE facility.**

ISOLDE has a history stretching over more than 30 years. During these years, the ISOLDE team has always been able to develop new and more intense isotope beams and the scientists have designed ever-more ingenious experiments for the facility. ISOLDE's unique ability to produce some 600 isotopes of 70 elements ensures it a place at the forefront of low-energy radioactive-beam research.

The heart of ISOLDE is the target-ion source system where a 1.0 or 1.4 GeV proton beam from CERN's PS-Booster accelerator strikes a target to produce a range of isotopes. These isotopes are ionised in an ion source close to the target and then accelerated. The beam goes through a magnet system, which extract the isotopes of interest based on their mass. These are finally delivered to experiments as a 60 keV beam. ISOLDE has always been able to produce extremely pure radioactive beams of a variety of species by combining a range of target materials with efficient and selective ion sources.

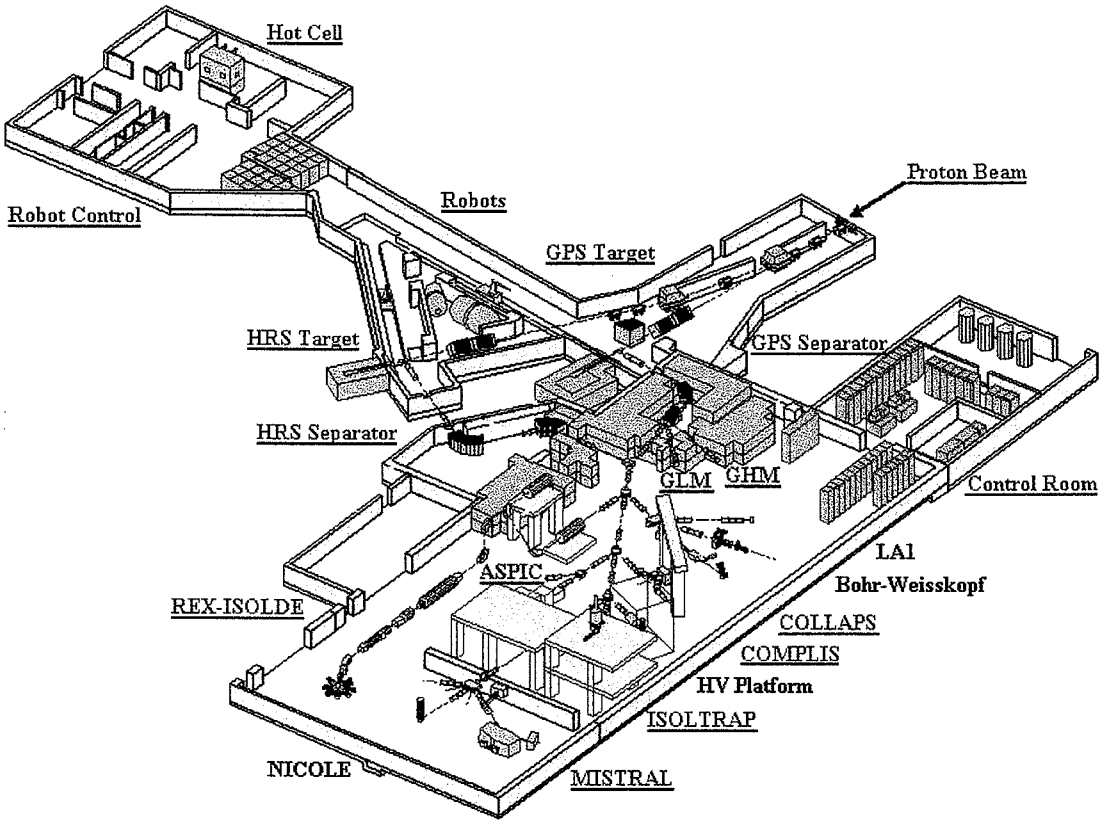



**Figure 1-5: Layout of the ISOLDE facility [2].**

ISOLDE actually has two isotope separators, the General Purpose Separator (GPS) and the High Resolution Separator (HRS). The GPS is designed to allow three different beams within a certain mass range (± 15% of the central mass) to be selected and delivered simultaneously. They are transferred into the experimental hall via three different beam lines (central mass, low mass and high mass beam line, the two latter ones denoted GLM and GHM respectively). The HRS can only deliver one beam, but with a higher resolution (M/ΔM up to 30,000 compared to the GPS's 2,400) [2].

New developments are made all the time in order to improve ISOLDE's capabilities and performance. An example of an important recent development is the Laser Ion Source (LIS). It works by shining a combination of three laser beams into the cloud of nuclei released from the ISOLDE target. The combination of photon energies selectively ionises the isotopes of interest. The two first laser pulses are tuned to excite the desired isotope (and other isotopes of the same element) by having just the correct photon energies to do so. Isotopes of other elements will not have the same excitation energies, and remain in their ground state. The third pulse has enough photon energy to ionise the exited isotopes, but not the other ones. This gives an unprecedented combination of both ionisation efficiency and selectivity.

### 1.3.1 REX-ISOLDE

A new experiment is currently being built at ISOLDE, called the REX-ISOLDE.

Until now, ISOLDE has concentrated its efforts on the study of radioactive nuclei at very low energies - less than 60 keV. A novel scheme for beam cooling, bunching, charge state breeding and subsequent acceleration will launch almost any isotope that is currently produced at ISOLDE into the 0.8-2.2 MeV per nucleon energy range. This will open up a completely new field of experiments with radioactive ion beams.

The idea is to convert a singly charged beam from the ISOLDE separator into a multiple-charged pulsed beam, which is suitable for acceleration, without losing too large a fraction of the painfully created radioactive nuclei. This requires advanced techniques. Initially the ISOLDE beam is trapped, cooled and bunched in a linear Penning trap. The bunched beam is then transported to an electron beam ion source, where it is confined while being bombarded with a strong electron current in order to reach higher charge states. The ions are then re-accelerated and separated according to charge state before reaching a Linac. The overall efficiency is estimated to be 10%.

The REX-ISOLDE is being developed and built by a large European collaboration and the first post-accelerated radioactive beams are expected during 2000.

## *1.4 Aim of this thesis*

Critical parts of the ISOLDE control system are getting old and outdated. To ensure safe and stable operation of the facility as well as easier maintenance, more modern standards are being gradually installed.

In the future, the widely used industrial fieldbus Profibus will replace a several of the existing interfaces between the equipment in the facility and the computers of the control system. Profibus will also be used in the REX-ISOLDE currently being built. The aim of the work on which this thesis is based, was to a large extent to implement this standard into the ISOLDE control system. One installation of Profibus had already been done at ISOLDE, but certain problems had been experienced with the use of this. In addition, the competence on how to install and use Profibus did no longer exist at ISOLDE. Finally, there was a desire to test other and newer variants of Profibus that required new control routines to be written, preferably in Visual Basic. All this has been done in this work.

The implementation of Profibus to the ISOLDE control system involved developing so-called Equipment Modules. The documentation of the ISOLDE control system is generally good, detailed and user-friendly. However, for some areas it is fragmented and partially non-existent. This is especially true regarding how to write these Equipment Modules. This thesis will provide an overview for how to create such modules, including some examples.

Chapter 2 will present the hardware and software systems that have been overviewed and used in this work. The most important of these systems are the ISOLDE Control System and the Profibus fieldbus standard. The chapter will try to provide persons without prior in-depth knowledge in these areas with enough information to understand the later chapters.

Chapter 3 presents the Visual Basic procedures written in this work for controlling a Profibus network. These are designed to be general enough to be re-used in other applications than those presented in this thesis.

In chapter 4, these procedures are used in so-called Equipment Modules for the ISOLDE control system. The chapter includes a general description of how to make such modules and two detailed examples of fully functioning Equipment Modules.

Chapter 5 and 6 concludes this thesis by presenting the results and findings of this work.

# 2 The technical background of this work

This chapter will present the basics of the software and hardware systems used in this work. It is meant to provide a platform for understanding the later chapters, and not as an exhaustive presentation of the various subjects.

The chapter contains information on the following subjects:

- Some advanced Visual Basic programming (section 2.1)
- The ISOLDE Control System (section 2.2)
- Profibus (section 2.3)
- Digital/analogue and analogue/digital converters (section 2.4)
- Quadrupoles (section 2.5)

## *2.1 Some advanced Visual Basic programming*

All of the programming in this work has been done in Visual Basic (VB) 5.0. Common object-oriented programming techniques including objects, methods, properties and events are presumed to be familiar to the reader, similar to the level in any introduction to VB programming. See for instance the "Visual Basic 5 for Windows for Dummies" [4].

However, a couple of the potentially more unfamiliar features utilised in the ISOLDE control system and this thesis will be treated briefly in the following.

### 2.1.1 Dynamic Link Library (.DLL) files

Dynamic Link Libraries contains commonly used procedures that can be shared among different programs. They are usually written in C++ or Pascal, but can be used by all programming languages.

To use a procedure in a DLL, a Declare statement must be included in the Visual Basic project. This statement contains the name of the library, the name of the procedure/function, the parameters that must be passed to it and the data type of the return value if it is a function that is being declared. Concerning the parameters, both its data type and whether it should be passed "by reference" or "by value" should be specified.

This is an example taken from a later section in this thesis:

```
Declare Sub writepackqbyte Lib "applicom" (Chan As Integer, Nes As Integer, nb As
   Integer, tadr As Long, Tabl As Byte, Stat As Integer)
```

This line of code contains the following information:

- 'Declare Sub' means that this is a procedure, not a function.
- 'writepackqbyte' is the name of the procedure.
- "applicom" is the name of the library.
- The information in the brackets is the names and data types of the parameters that should be passed to this procedure. Since the ByVal keyword is not included before any of the parameter names, they are all passed by reference.

Another example will show how a function is declared:

```
Declare Function dpn_init Lib "dplib.dll" (dpn_intf As dpn_interface) As Integer
```

The 'Declare Sub' statement has changed to 'Declare Function' and after the brackets the words 'As Integer' are included to tell the data type of the function's return value. It takes only one parameter, which is of a custom data type called dpn_interface and passed by reference (see section 3.1.1.2 on page 33 for details on this data type).

After having been declared, a function or a procedure from a DLL can be used just as if it was included in one's own program. This is an example of how the writepackqbyte procedure declared above is used:

```
Call writepackqbyte(CardNumber, SlaveNo, 2, offset, OutData(1), cc)
```

The parameters have got different names than in the declaration, and one of them is passed as a number. 'OutData(1)' means that OutData is an array, and that only the first element is passed to the procedure. The rest of the array can be found by the procedure in the computer's memory, since it is passed by reference.

In order for Visual Basic to find a .DLL file, it must be put in a directory included in the "PATH" environment variable.

### 2.1.2 ActiveX controls

An ActiveX control is a subprogram that can be re-used in other programs in a simple way. It is a part of Microsoft's Component Object Model (COM), which is a technology for transparently transferring encapsulated data across boundaries such as separate modules, threads, processes or machines. A more exact definition is almost impossible to give [5], but the most important point for the use of ActiveX controls in this work is explained below.

In this thesis, ActiveX controls show up as an important part of the ISOLDE Control System. This is both because most of the code is placed in different ActiveX controls and because they provide an easy way to attach different parts of the system to each other.

Some of the components created during the work of this thesis are also ActiveX controls. These are the components that control the hardware, see chapter 4 starting on page 45.

An ActiveX control is used in a VB project by adding it to the list of components in the project (menu selection *Project, Properties...*). The objects in the ActiveX then

appears as icons in the Visual Basic Toolbox. An object is made available to the current project by selecting its icon and then drawing a so-called instance of it on a form in the project by clicking and dragging on the desired position. Several instances of the same object can be drawn, and be referred to by their unique name.

This object then has methods and properties that are used for retrieving data from or transferring data to another part of the ISOLDE control system.

### 2.1.3 Timing

The standard `Timer` function in Visual Basic gives the number of seconds elapsed since midnight with in a 4 byte floating point variable. However, as the precision of this function is known not to be very good [5], another technique is used in this thesis for measuring running times of parts of the code. This involves the Windows API* function `QueryPerformanceCounter`, which makes it possible to measure times with millisecond precision or better.

The Visual Basic project must include the Windows API as a reference, and the following procedures should be added to any module:

```
Private secFreq As Currency

Sub ProfileStart(secStart As Currency)
    If secFreq = 0 Then QueryPerformanceFrequency secFreq
    QueryPerformanceCounter secStart
End Sub

Sub ProfileStop(secStart As Currency, secTiming As Currency)
    QueryPerformanceCounter secTiming
    If secFreq = 0 Then
        secTiming = 0 ' Handle no high-resolution timer
    Else
        secTiming = (secTiming - secStart) / secFreq
    End If
End Sub
```

The time it takes between two points in the code can then be measured like this inside a procedure:

```
Dim StartTime As Currency, TimeTaken As Currency

ProfileStart StartTime                'Starts timer
  :
(code to be timed)
  :
ProfileStop StartTime, TimeTaken      'Stops timer
```

The variable `TimeTaken` now contains the number of microseconds between the two procedure calls.

## 2.2 The ISOLDE Control System

The ISOLDE control system has more than 1500 control channels for controlling the different parts of the isotope separator. These parts are pumps, valves, benders, lenses, magnets, scanners, Faraday cups and so on, and each of them is called one piece of equipment with their own name. The name will usually tell in which part of the

* Application Program Interface, a set of routines, protocols, and tools for building software applications.

machine the equipment is located and which type of equipment it is. In addition, it can have a number to keep similar equipment apart. A quadrupole lens (see section 2.5) can have the name GPS.QP30 revealing that it is placed in the GPS, that it is a quadrupole (QP) and that it has the identity number 30. GHM.BE10 is a bender situated in the high mass beam line of the GPS.

The task of the control system is to provide easy access to all these parts that ISOLDE is made up of. The current control system was developped by A. Pace, G. Shering and I. Deloose from the control group of CERN's PS Division and became operative in 1992. In the following years, the system evolved into today's architecture, with TCP/IP as the communication protocol. It is tailor-made with high modularity and easy access as its main advantages. This was appreciated during this work.

### 2.2.1 Hardware architecture

The control system of the isotope separators and beam lines are based on personal computers (Intel 80386 or higher microprocessor running under MS-DOS or Microsoft Windows). Network wide distributed front-end computers, which access the hardware for controls and measurements, are controlled by PC-consoles via a local area network with a PC file server used as database. This local network is part of the CERN-wide Ethernet network, which means that FECs and consoles in principle can be placed anywhere at CERN. This is shown in Figure 2-1 below.

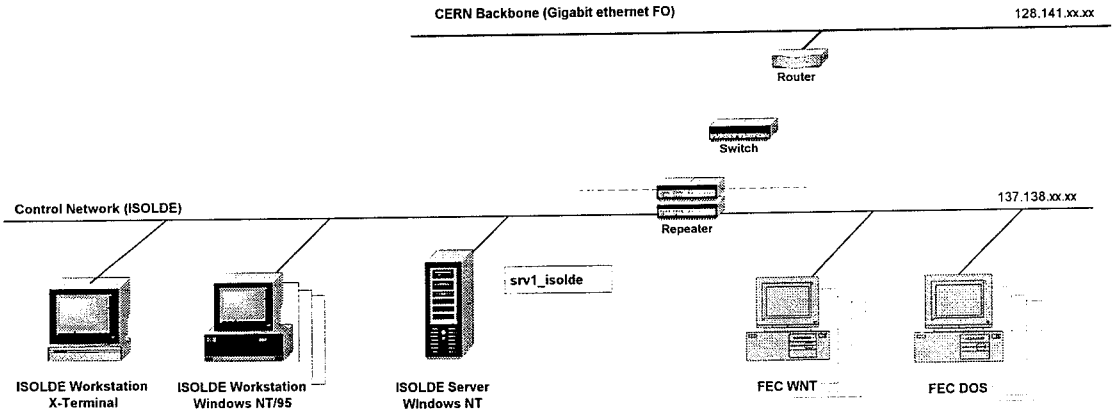



**Figure 2-1: The ISOLDE Control Network.**

One Front-end Computer (FEC) can be connected to dozens of pieces of equipment, and one console can control the same amount of equipment simultaneously via small control panel applications.

The FECs use a variety of interfaces to communicate with the equipment. Among these are GPIB[†], CAMAC[‡] and various I/O cards with analogue, digital and RS232 inputs and outputs [6]. The strategy for the future is to gradually diminish this

[†] General Purpose Interface Bus. The GPIB bus is used for controlling hardware from a computer. It is an 8-bit parallel bus designed mainly for measuring equipment, and allows 15 intelligent devices to share a single bus, with the slowest device participating in the control and data transfer handshakes to drive the speed of the transaction. The maximum data rate is about one megabit per second.

[‡] Computer Automated Measurement And Control. This is an IEEE-standard (583), modular, high-performance, realtime data acquisition and control system concept. Born in 1969 and widely used since, but now considered to be outdated.

multitude of interfaces and use an industrial fieldbus as a standard, as explained in the first chapter (section 1.4). The fieldbus that has been chosen (Profibus) will be introduced in section 2.3.

The ISOLDE server is used as a database where all equipment data are stored (see section 2.2.2.1). In addition, the software parts of the control system reside here.

### 2.2.2 The software architecture

The software architecture of the ISOLDE control system is rather complex, as shown in Figure 2-2.

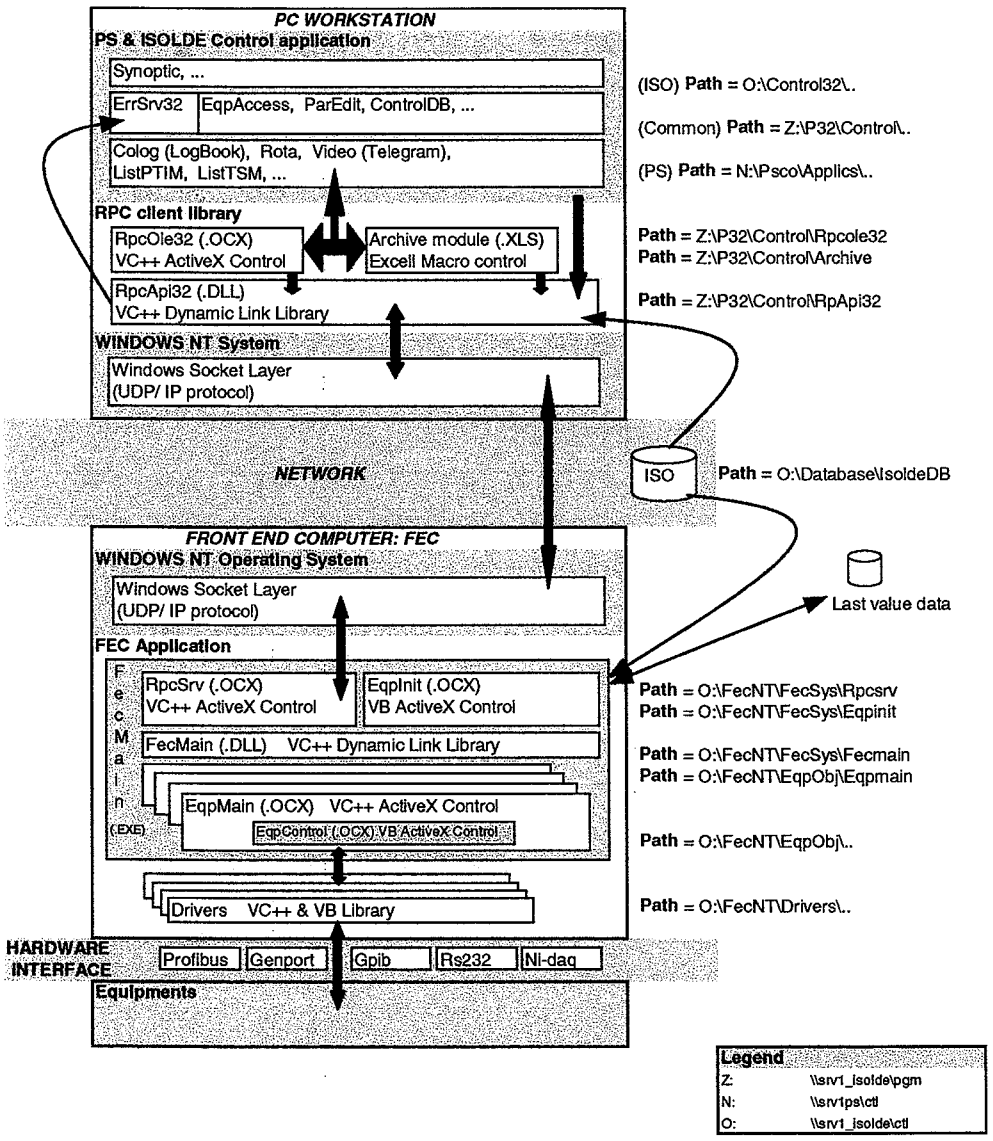



**Figure 2-2: The software architecture of the ISOLDE Control System.**

Starting from the bottom of the figure above, the work of this thesis has consisted of implementing the Profibus hardware interface, creating so-called Equipment Modules ("EqpControl (.OCX) VB ActiveX Control" in the figure) and creating graphical user interfaces ("Synoptic, ..." topmost in the figure).

The modularity of the control system implies that the details of the whole system do not have to be known in order to create a user interface or an Equipment Module. Actually, only three of all the different parts of the control system have been directly accessed in this work, namely the EqpMain ActiveX control, the ISOLDE database and the RpcOle32 object in the RpcOle32 ActiveX control. Some details of these 'interface' parts of the control system will be explained in the following.

### *2.2.2.1 The ISOLDE Database*

The database is used for storing all necessary data about the equipment in the ISOLDE facility, including front-end computers, interfaces, input and output channel numbers, data formats and so on. These data are being read by the FECs during their start-up and by the applications trying to access the equipment (e.g. user interfaces).

Until 1997, all the data concerning the FECs and the equipment was stored as MS Excel tables. In addition, there existed certain general lists such as properties for each class of equipment, name and properties of the FECs and so on, but no relations between these informations. In 1997, Ivan Deloose therefore made a relation database in MS Access. The layout of this database is shown in Figure 2-3.

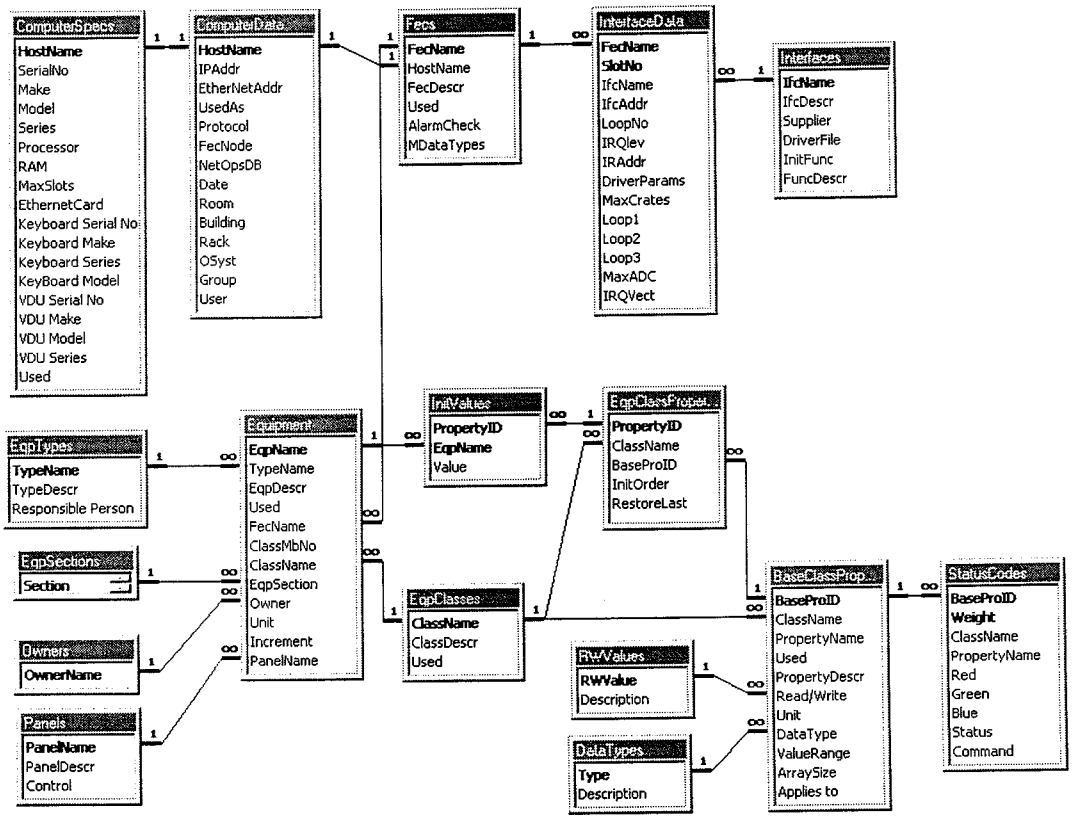



**Figure 2-3: The layout of the ISOLDE database.**

The database is edited with a special editor made by Anne-Heidi Evensen Mills. This editor controls the access to the database and checks the data being put into it.

More detail about the data entered into the database when adding new equipment or software to the control system is explained in section 4.2.

#### *2.2.2.2 The EqpMain ActiveX control*

This ActiveX control hides all common equipment programming complexity and is written in C++ for performance and multi-threading reasons. It can be considered as the interface between the core of the control system and the user-written Equipment Modules.

When called, EqpMain contains a dynamic data structure of all properties and members of an equipment class. This means that every property defined in the database is automatically registered and accessible through the `EqpMain.Propertydata` property. Only equipment specific code has to be written in an Equipment Control.

Also EqpMain will save the most recent property data into a local text file table. This can be used later when the FEC reboots and needs to recover the latest settings of the equipment.

Another thread can be started on request of the Equipment Module, used for asynchronous data acquisition. The idea here is that EqpMain fires an event to the Equipment Module (`EqpMain_Callback()`) for every registered equipment member. The implementation of this event is located inside the Equipment Module and has to contain the necessary code for the hardware access and data storage in the property buffers. See section 4.4.3.3 on page 52 for an example.

#### *2.2.2.3 The FecMain application*

When a FEC is starting up and running, it is this application that is executed. It can be considered a component manager and communicator, dispatching messages between several controls.

The window of the application is divided into two main parts, called "Initialisation Requests" and "RPC Requests". They represent one ActiveX module each, namely EQPINIT.OCX and RPCSRC.OCX.

The former connects the NT FEC to the database and return information to FecMain about hardware device drivers and equipment control to be loaded. In its window, the information about the different steps of the initialisation appears as they happen.

The latter makes the connection between the clients (console PCs) on the network and FecMain. It handles cold and hot links (see next section). The complete data subscription mechanism is handled inside this control, and during the running of the FEC, its window is constantly updated with information about the information exchange going on.

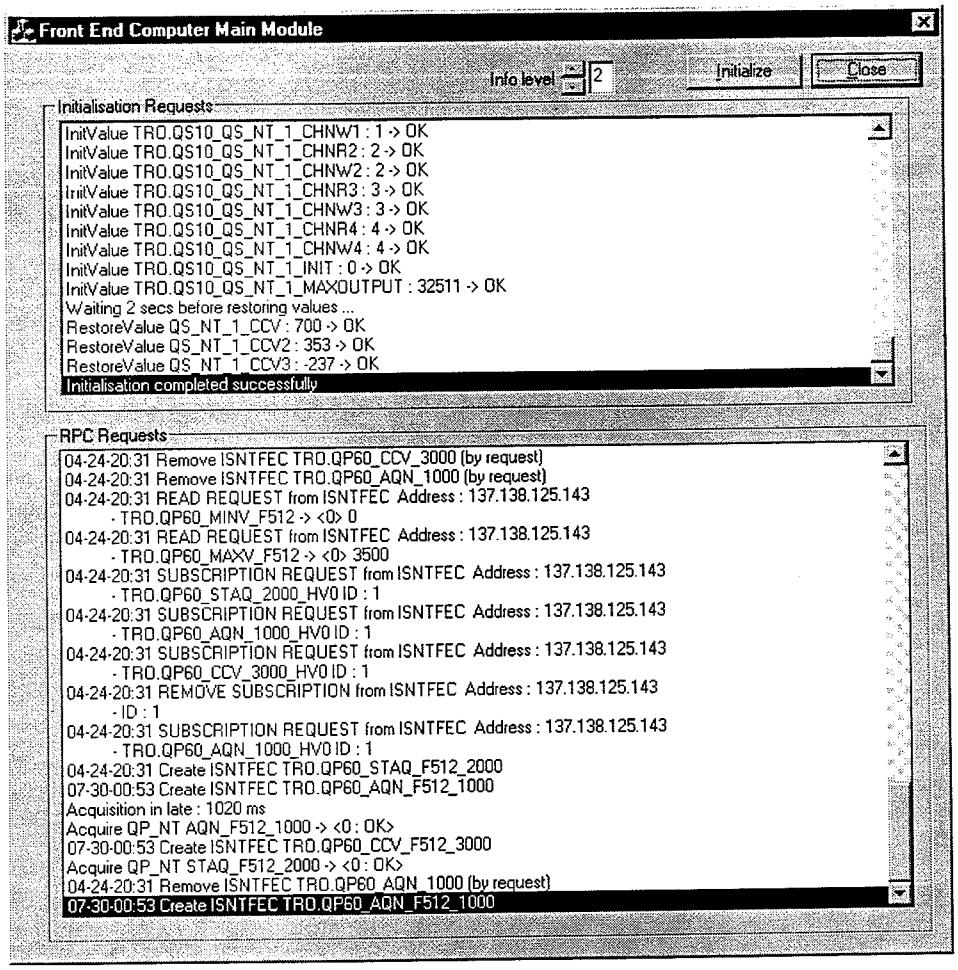



**Figure 2-4 : The appearance of the FecMain applicatin. Notice the two main windows and the 'Info level' indicator.**

A feature has recently been added to this application: the possibility to choose the amount of information shown in the "RPC Requests" windows of the application. This is done because most of the load on the FEC's processor comes from updating the screen every time a call is made to any equipment connected to it. Info level '2' is the standard, and all information is shown. Info level '0' means that no information is shown, and a considerable amount of time is saved for each call to the FEC.

#### *2.2.2.4 The RpcOle32 object*

The RpcOle32 object [7] is meant to provide to the user a simple way of accessing the control system without any advanced programming. Just as EqpMain is the interface between the control system and the Equipment Modules, this object is the interface between the control system and user applications such as control panels.

RpcOle32 is an object included in a C++ ActiveX control that can easily inserted into any Visual Basic project. Then, after drawing an instance of it on one's own Visual Basic object and specifying the equipment name, the properties of any equipment in the ISOLDE database is accessible as properties of this object.

Its features include the possibility to read and write property values and to establish hotlinks for automatic data update.

This is used when making graphical user interfaces as shown in section 4.4.4 on page 55.

## 2.3 Profibus

As most of the work with this thesis has been to implement Profibus to the control system, a thorough description of the Profibus basics is included in the following.

### 2.3.1 Introduction

Profibus (Process Field Bus) is a standard for a heterogeneous network of automation components. Heterogeneous means that equipment from different vendors that comply with the standard can be used. It is one of eight fieldbus systems included in the European Fieldbus Standard EN 50170.

Fieldbuses are a special form of local area network dedicated to applications in the field of data acquisition and the control of sensors and actuators in machines or on the factory floor. Fieldbuses typically operate on low cost twisted pair cables. Unlike traditional networks, such as Ethernet, where performance is measured in throughput when transferring large data blocks, fieldbuses are optimised for the exchange of short point-to-point status and command messages.

A fieldbus like Profibus has numerous advantages to older automation systems currently in use at ISOLDE. The most important are low wiring costs since up to 126 units (slaves) can be connected to one twisted-pair cable, the fact that equipment from different vendors can be used on one bus and the simplification of maintenance achieved from fewer interfaces being in use. It is also economical in the way that the distributed devices (slaves) have been designed to be as simple and cheap as possible.

Figure 2-5 below shows the layout of a typical Profibus network as used in this work. It is controlled by a computer (PC) with a special Profibus card installed. This is called the master. The cable is then connected from this master to all the units that are to be controlled by this computer. These units are called slaves, and consist of a buscoupler and one or more input and/or output modules. All the I/O modules used in the work with this thesis were DACs and ADCs[§] operating in the 0-10 V range. Depending on the type of buscoupler being used, one slave can have up to 128 input channels and 128 output channels. (One module usually has two, four or eight channels.)

Profibus is one of three fieldbuses recommended by The Working Group on Fieldbuses at CERN in 1996 [8].

[§] Digital to analogue and analogue to digital converters. See section 2.4

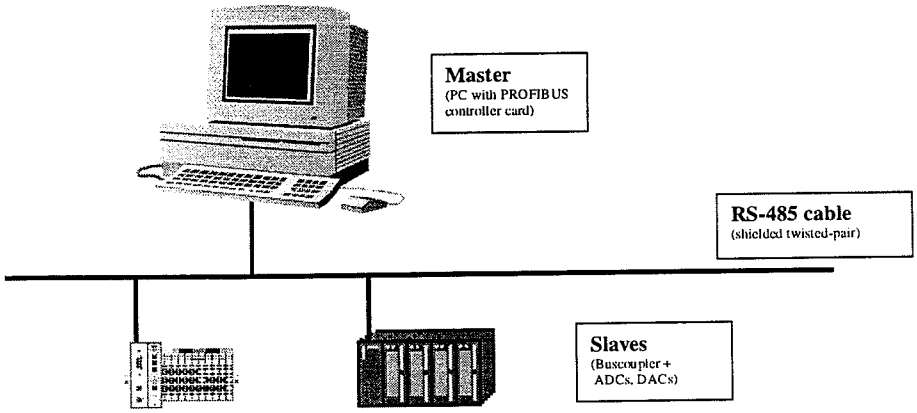



**Figure 2-5: Schematic layout of a typical PROFIBUS network.**

### 2.3.2 The ISO/OSI reference model and the technical specifications of Profibus

Today all commercial automation and communication systems, including Profibus, are based on the ISO/OSI reference model [9,10,11,12]. It provides a framework for standardising communication procedures in seven layers, see Figure 2-6. This framework shall ensure compatibility between equipment from different vendors, high reliability in data transmission and easy use and configuration of the communication systems.

| | The Layers | Their Tasks |
|---|---|---|
| 7 | Application layer | Provides uniform access to specific services. |
| 6 | Presentation layer | Establishes transfer syntax. |
| 5 | Session layer | Establishes, maintains, terminates communication sessions |
| 4 | Transport layer | Transfers "raw" information reliably. |
| 3 | Network layer | Provides network connections. |
| 2 | Data link layer | Establishes and terminates links. |
| 1 | Physical layer | Regulates physical connections for bit transfer. |

**Figure 2-6: The seven layers of the ISO/OSI reference model.**

In Profibus, only layers 1, 2 and 7 are implemented for reasons of performance and to reduce costs.

Layer 1 specifies that the cable can be a shielded twisted-pair RS-485, an optical fibre or transmission in accordance with IEC 1158-2 for use in hazardous areas. In this work, only the RS-485 has been used, but when Profibus is installed more extensively in the ISOLDE hall, the optical fibre will also be used. The RS-485 provides for a half duplex communication, since a station cannot simultaneously transmit and receive data streams. Both wires of the twisted-pair are on a positive potential compared to ground, with the polarity of the voltage difference between the two wires determining

whether the signal is a 0 or a 1. It is therefore important to keep track of which wire is which, normally designated "A" and "B".

The data link layer, or layer 2, is responsible for several functions. It structures the information into well-defined groups, identified as "frames" or messages, handles identification of source and destination stations on the network, and provides for error-free transmission of a message from source to destination stations. For Profibus, this layer is implemented as Fieldbus Data Link (FDL). This specifies for instance that the messages are sent with a Hamming Distance of HD=4, which means that up to three bit errors can be detected and corrected [11]. This implies that PROFIBUS guarantees a high secure protection against transmission errors.

The application layer provides communication services directly to the user application. This layer is implemented for some variants of Profibus, but not for the DP-Profibus that has been used in this work. For DP-Profibus this layer is skipped, and the user is given access to the lower layers through a dynamic link library (DLL-file). This provides the necessary functions for interaction with the network.

### 2.3.3 DP-Profibus

DP stands for Distributed Periphery and this variant of Profibus is the performance optimised one. It is specifically dedicated to time-critical communication between automation systems and the distributed peripherals (slaves).

Compared to other protocols, the DP-Profibus have the following characteristics [10]:

- Central control by one master.
- High data throughput thanks to a simple transmission protocol.
- Cyclic transmission of the process image in the input and output direction.
- Simple, cost-effective network attachment.
- Data transmission with the option of twisted-pair (RS 485) or fiber-optic cables.
- Detection of errors with on-line diagnostics.

As briefly explained in section 2.3.1, the masters used in this work are computers (PCs) with Profibus Communications Processors (CPs) installed. The master controls communication with the distributed I/O devices (slaves) and handles the central functions of a DP master class 1 according to EN 50 170. This involves the following functions:

- Parameter assignment/configuration of the DP slaves.
- Initialisation of the DP system.
- Cyclic data transfer to the DP slaves.
- Monitoring the DP slaves.

The data exchange with the slaves is mainly cyclic. The central controller (master) cyclically reads the input information from the slaves and cyclically writes the output information to the slaves. Cyclic means here that one telegram is sent from slave to slave according to slave number. One slave answers before the telegram is sent to the next slave. The bus cycle time should be shorter than the program cycle time of the central automation system, which for many applications is approximately 10 ms. In

this work a cycle time of less than 2 ms has been used. Longer cycle times only occurs for networks with a large number of slaves operating on low baud rates (15 slaves and 500kBit/s gives a cycle time of about 10 ms) [12].

Additional features of DP-Profibus include cyclic (asynchronous) user data transmission and powerful functions for diagnostics and commissioning.

#### *2.3.3.1 The DP programming interface*

When programming an application for DP-Profibus, one makes use of the DP library. This communicates with the Profibus CP through a driver, and provides access to the bus. The different components are shown in Figure 2-7.

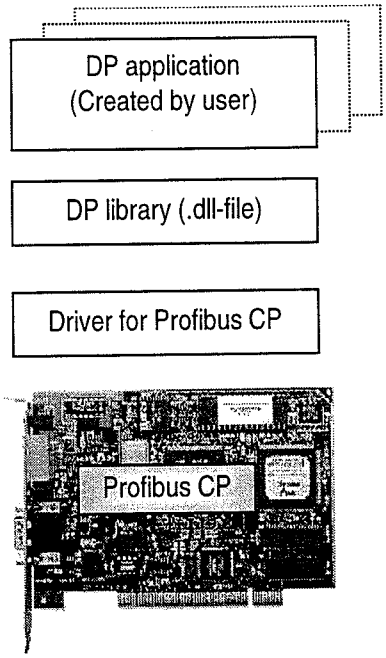



**Figure 2-7: The software and hardware components of DP-Profibus.**

Equipment from different vendors have different programming interfaces, with different functions in the .dll-library, different ways of handling multiple DP applications and so on.

In this work, all DP applications are in form of a Visual Basic ActiveX application (OCX).

More details about making DP applications can be found in chapter 3.

### 2.3.4 Installing and operating a DP-Profibus network

The previous sections have explained the basics about Profibus. That is all theory, however. In order to make a Profibus network work in real life, certain actions have to be taken. The following will explain this briefly, so that later sections regarding Profibus can more easily be understood. For a more complete description of this, please refer to Appendix A or a Profibus manual [10, 13, 14].

#### *2.3.4.1 Installation of the bus*

The installation of a DP-Profibus network basically consists of installing a communication processor (CP) card in the PC together with the software provided, and then connecting a Profibus cable between the CP and the different slaves in the network. Of course, the slaves also have to be connected to the equipment they are to control and monitor, but that does not have to do with the Profibus.

#### *2.3.4.2 Configuration of the bus*

The specifications of the Profibus network has to be put into a database stored on the computer. This is usually done with a configuration program included in the software package of the CP, such as COM PROFIBUS (Siemens) or PCCONF (applicom).

Among the information that has to be specified is the settings of the CP (baud rate etc.) and the specifications of all the slaves on the bus (slave number, type, I/O channels). This is done more or less automatic with the help of the configuration program and data files about the slaves (gsd-files).

#### *2.3.4.3 Initialising and running the bus*

In the initialisation process, the database made in the configuration of the bus is loaded. It is checked that the CP is operational, and the master starts to send data on the bus. The data sent is a general broadcast that the bus is operative, together with relevant output data to the initialised slaves. No answer from the slaves is needed.

The output data sent can be zero, another fixed value or the last output value stored in the CP, depending on the type of CP and slave, and what was specified in the configuration of the slaves.

When a read or write call is made to the master from a DP application, the database is read and the correct data are included in a message sent out on the bus. The correct slave then answers if it is present and operational, and returns a message to the master. The CP finally answers to the DP application whether the request was successful or not, together with any input data.

### 2.3.5 The Profibus set-ups used in this work

All the work regarding Profibus in this thesis has been done on a small Profibus network consisting of one master and two slaves.

Two different masters were tested, both were PCs with a Communication Processor (CP) in stalled. The two CPs were from different vendors, one CP 5412 from Simatic Net produced by Siemens and one *applicom*® PCI$^{1500PFB}$ from applicom international. The programming of these is discussed in chapter 3.

The two slaves, given slave numbers 4 and 5, were also of different types. Their specifications are given in Table 2-1 below. For general information about digital/analogue and analogue/digital converters, see next section.

**Table 2-1: Specifications of the Profibus slaves used in the development of the Equipment Modules [15, 16].**

| | | | |
|---|---|---|---|
| **Slave number** | | 4 | 5 |
| **Producer** | | Simatic | Wago |
| **Number of output channels installed** | | 20 | 2 |
| **Number of input channels installed** | | 24 | 4 |
| **Range of digital values (for 0 – 10 V in/out)** | | 0 .. 27648 + overrange to 32511 | 0 .. 32767 |
| Buscoupler | **Model** | ET 200M (model IM153-1) | 750-303 |
| | **Dimensions (mm)** | 40 × 125 × 120 | 51 × 65 × 100 |
| | **Baud rate** | 9.6 kBaud .. 12 MBaud | 9.6 kBaud .. 12 MBaud |
| | **Response time** | 1.0 ms (typ.) | 1.0 ms (typ.) |
| | **Maximum number of I/O modules** | 8 | 64 |
| Output Modules (DACs) | **Model** | 332-5HD01-0AB0 (4 x 12 bit AO) | 750-550 (2 x 12 bit AO) |
| | **Dimensions (mm)** | 40 × 125 ×120 | 12 × 64 × 100 |
| | **Output range** | Multiple, incl. 0 - 10 V | 0 – 10 V |
| | **Load impedance** | Min. 1 kΩ | > 5 kΩ |
| | **Injection of substitute values** | Yes, programmable | No |
| Input Modules (ADCs) | **Model** | 331-7KF01-0AB0 (8 × 12 bit AI) | 750-468 (4 × 12 bit AI) |
| | **Dimensions (mm)** | 40 × 125 ×120 | 12 × 64 × 100 |
| | **Input range** | Multiple, incl. ± 10 V | 0 – 10 V |
| | **Conversion time (typ.)** | 200 ms | 2 ms |
| | **Internal resistance** | 100 kΩ for the ± 10 V range | 133 kΩ typ. |

The differences between the different types of slaves and masters will be discussed in section 5.2.

Both set-ups (masters) were configured with about 2 ms cycle time on the bus.

## 2.4 Digital/analogue and analogue/digital converters

A few words about digital-to-analogue converters (DACs) and analogue-to-digital converters (ADCs) [11] should be said, as it is these that transfer the information from the Profibus to the actual equipment.

### 2.4.1 DACs

These are the output modules being used. It converts a digital number to an analogue signal proportional to the input number.

One of the most important characteristic properties for a DAC is the resolution. It is measured in bits and indicates the number of steps between the lowest and the highest output value. A resolution of 12 bits means that there are $2^{12} = 4096$ steps. A 16 bits resolution gives 65536 steps.

In general, a higher resolution means a more precise output signal, but often the noise suppression and error limits are the limiting factors. One step in a 16 bits DAC represents only 0.00153 % of the output range, whereas other factors can limit the accuracy to about ± 0.5 % of the output range [16].

The analogue output signal being used in this work is a 0-10 V DC voltage.

### 2.4.2 ADCs

These are the input modules of the slaves. It converts an analogue input signal to a digital number proportional to the input signal.

Whereas digital-to-analogue conversion is barely a multiplication of a reference voltage, going the opposite way is a bit more complicated. There exist different techniques for this conversion. The ADCs used in this work are based on two different principles: successive approximation and integration.

#### *2.4.2.1 Successive approximation converters*

The principle of this type of converter is shown in Figure 2-8 below.

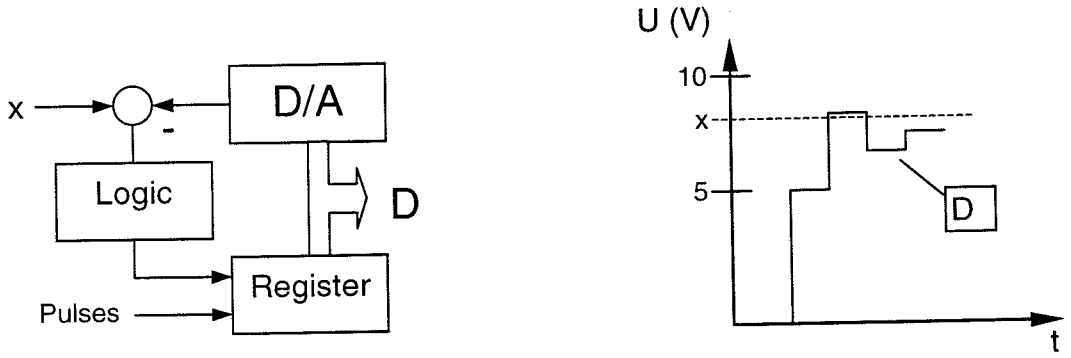



**Figure 2-8: Successive approximation A/D converter. *x* is the analogue input signal and *D* is the digital output signal.**

The idea is to use a DAC in a feedback loop, and then try bit by bit until the correct digital output value is found. First, all bits are set to zero. The most significant bit is then set, and if the output signal is still less than the input signal, it is kept and otherwise rejected and set to zero again. On the next pulse, the next bit is tried and so on.

The conversion time is always the number of bits multiplied by the time it takes to test each of them, no matter where in the range the input signal is.

Successive approximation converters can be very fast, down to 1 µs for 12 bits.

### *2.4.2.2 Integrating A/D converters*

The principle of this type of converter is shown in Figure 2-9 below.

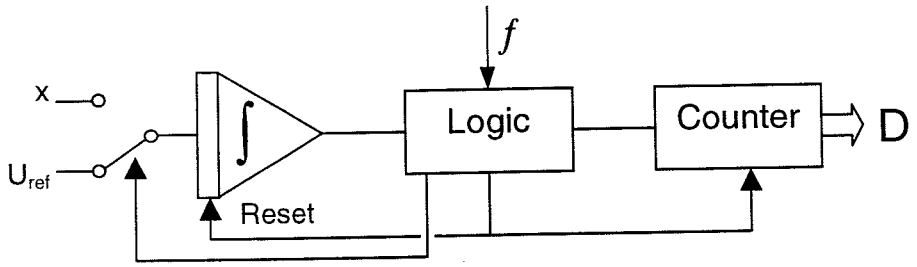



**Figure 2-9: An integrating A/D converter. $x$ is the input signal, $U_{ref}$ is a reference voltage, $f$ is a counting frequency and $D$ is the digital output signal.**

A conversion can be separated into two parts:

1. First, the counter and integrator are reset to zero and the integrator is connected to the input signal. The integration starts and the counter counts with the frequency $f$ until it has counted $2^n$ pulses. At this point the integration is stopped, and its output signal is now
$$u_i = \int_0^{2^n/f} x dt$$
2. The input selector is changed to the reference voltage, which is the same as the upper limit of converter's input, but with the opposite sign. The integrator and the counter are started, but this time the integrator starts at the value $u_i$ and integrates downwards. The counter starts at zero and counts until the integrator reaches zero. The output number from the counter is the digital output value. This is because we at this point have
$$0 = \int_0^{2^n/f} x dt - U_{ref} \int_0^{N/f} dt$$
$$N = \frac{f \int_0^{2^n/f} x dt}{U_{ref}},$$

where N is the output of the counter when the integrator reaches zero.

The frequency is part of both integrals, and is hence not critical as long as it is stable. The critical factor is the reference voltage.

The resolution of an ADC is important, as for DACs. However, the better the resolution, the longer the conversion time. Remember that the counter counts to its maximum limit during the first integration. The relatively long conversion time is the biggest drawback of integrating converters.

However, this can be used as an advantage. Since the conversion time is rather long in the first place, the integration time can be chosen to be a multiple of 20 ms. This is for reason of noise suppression, since 20 ms is the period of the 50 Hz noise from power lines. This is based on the fact the integral over a whole period of a sinusoidal signal is zero.

Integrating ADCs are characterised by good accuracy, low price and good noise suppression.

The analogue output signal being used in this work is a 0-10 V DC voltage.

## 2.5 Quadrupoles

In this work, Equipment Modules for quadrupoles [17] have been created. Quadrupoles are used for focusing of the ion beams. At ISOLDE, some of them are also used for steering of the ion beam, due to limited space in the beam lines.

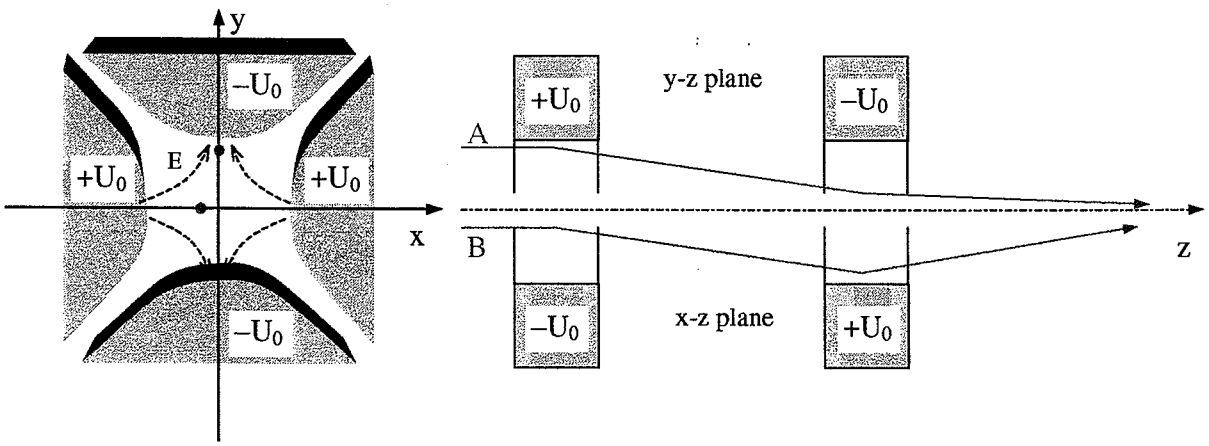
The basic principles of these will be explained below.

### 2.5.1 Focusing quadrupoles

A schematic representation of an ordinary quadrupole is shown in Figure 2-10. With the potential signs in the figure the ideal electrical quadrupole field will be:

$E_x = -E_0 x/r_0 \qquad E_y = -E_0 y/r_0,$

where $r_0$ indicates the distance from origin to the electrode surface in the x and y directions, and $E_0$ is the electrical field on the electrode surface.




a) A quadrupole lens. $U_0$ is the *quadrupole voltage.*

b) A quadrupole lens pair with two particle trajectories, viewed in the y-z and x-z planes.

**Figure 2-10: An electrostatic quadrupole lens and two typical trajectories.**

The field is proportional to the distance from the axis of the system, but of opposing signs in the *x* and *y* directions. One quadrupole lens alone will focus in one direction and defocus in the other. A useful quadrupole will therefore consist of two such lenses, polarised 90° with respect to each other. This creates a focusing-defocusing pair in one direction and a defocusing-focusing pair in the other. The net result is a lens that is strongly focusing in all directions.

Quadrupoles are widely used in machines where high-energy particle beams have to be guided and collimated; there are for instance 72 quadrupoles in ISOLDE [18].

### 2.5.2 Steering quadrupoles

A steering quadrupole is very similar to an ordinary quadrupole, except that the electrode potentials are applied with an imbalance. This results in a simultaneous focusing and deflection of the ion beam and is used for steering the beam through the evacuated pipes, as the name indicates.

A steering quadrupole looks just like a focusing quadrupole, with the difference lying in the applied voltages. In principle, an ordinary quadrupole voltage is applied and then a horizontal and/or vertical steering voltage is added. The resulting total potential of each electrode is shown in Figure 2-11.

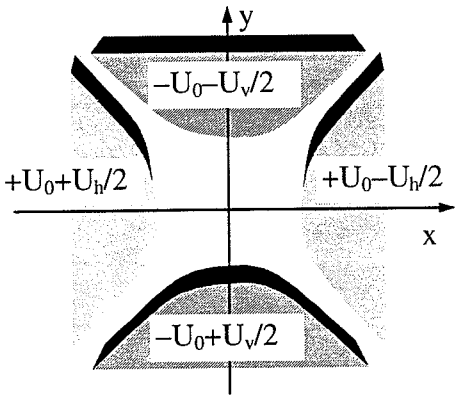



**Figure 2-11: A steering quadrupole with electrode potentials indicated. $U_0$ is the quadrupole voltage and $U_v$ and $U_h$ are the vertical and horizontal steering voltages.**

To keep the beam aberrations at a permissible level, the steering voltages that can be applied are restricted. At ISOLDE, the maximum steering voltage is usually set to 500 Volts.

There are 35 steering quadrupoles at ISOLDE.

### 2.5.3 The REX quadrupoles

In the ISOLDE-REX some quadrupoles will be used almost like ordinary focusing quadrupoles, with the exception that two separate voltages are applied: one for the upper and lower electrodes and one for the left and right electrodes. This gives a focusing quadrupole, but with possibly different focusing power in the two axis directions. The quadrupole can therefore also shape the beam if the two voltages are not the same.

# 3 Controlling Profibus with Visual Basic

In the process of making Equipment Modules using Profibus as hardware interface, there was a need for writing new procedures for controlling the bus. The existing Equipment Module using Profibus made use of a driver written in C by Ivan Deloose. It was desirable to have full control of how the access to the bus was made both in order to see how the Profibus performs and to be able to add new procedures if necessary. In addition, this driver was only written for the Simatic net DP-5412 and new procedures had to be written for Applicom anyway since the two variants are controlled in different ways by different functions.

This chapter will present the result of this work, with the rewritten procedures for making the actual hardware calls being treated in detail. This will show how to control a Profibus network using Visual Basic, with either a Sinec CP 5412 (A2) or applicom® PCI$^{1500PFB}$ card as a master. All functions are made as general and easy to use as possible. The following procedures are implemented:

- InitDP for initialising the bus
- ResetDP for resetting the bus
- ReadData for reading data from a given input channel
- SendData for sending data to a given output channel
- ReadOutput for checking the current state of a given output channel

To include Profibus functionality to a Visual Basic project it is sufficient to add two modules, one header file for the .dll library and one module containing the functions listed above, the latter one named ProfiProcedures.

## *3.1 Simatic net DP-5412*

The Simatic net DP-5412 [19] uses a CP 5412 (A2) card as a master and a library named dplib.dll for control of this. In the described Visual Basic program, the necessary declarations for using this library are placed in a module called dplib (dplib.bas). The module containing the actual procedures is called ProfiProcedures (Profi.bas). This chapter contains a detailed description of the contents of these two modules. From a CERN computer, these modules can be found in the folder \\SRV1_ISOLDE\CTL\FecNT\EqpObj\QP_NT\Src (where \\SRV1_ISOLDE\CTL\ is usually mapped as O:\).

### 3.1.1 The declarations module (Dplib)

This module have three different parts: The declaration of constants including error codes, the type declarations of the data structure necessary for passing data to the functions in the dplib.dll library and the "headers" for these functions

### *3.1.1.1 The constants*

The following constants are declared:

This is the maximum number of elements in the `user_data()` array.

```
Public Const MAX_DATA_LEN = 255


'************************************/
'* Highest slave and channel number */
'************************************/
Public Const MAX_SLV_NO = 125
Public Const MAX_CHN_NO = 128

'****************************************/
'* Defines for "reference_struct.access" */
'****************************************/
Public Const DPN_SYS_CENTRAL = 32
Public Const DPN_SYS_NOT_CENTRAL = 16
Public Const DPN_ROLE_CENTRAL = 128
Public Const DPN_ROLE_NOT_CENTRAL = 64

'**********************************************/
'* Defines for function "dpn_read_slv_par()" */
'**********************************************/
Public Const DPN_SLV_PARA_TYP = 0
Public Const DPN_SLV_PARA_PRM_DATA = 1
Public Const DPN_SLV_PARA_CFG_DATA = 2
Public Const DPN_SLV_PARA_ADD_TAB = 3
Public Const DPN_SLV_PARA_USER_DATA = 4

'**********************************************'**/
'* Defines for function "dpn_set_slave_state" */
'**********************************************/
Public Const DPN_SLV_ACTIVATE = 128
Public Const DPN_SLV_DEACTIVATE = 0

'**********************************************/
'* Defines for function "dpn_read_cfg"        */
'**********************************************/
Public Const DPN_CFG_NO_SLV = 255
Public Const DPN_CFG_NORM = 0
Public Const DPN_CFG_ET200_U = 1
Public Const DPN_CFG_ET200K_B = 2
Public Const DPN_CFG_ET200_SPM = 3

'**********************************************/
'* Defines for structure element "sys_state"  */
'**********************************************/
Public Const DPN_SYS_OFFLINE = 0
Public Const DPN_SYS_STOP = 64
Public Const DPN_SYS_CLEAR = 128
Public Const DPN_SYS_OPERATE = 192

'**********************************************/
'* Defines for structure element "slv_state"  */
'**********************************************/
Public Const DPN_SLV_STAT_OFFLINE = 0
Public Const DPN_SLV_STAT_NOT_ACTIVE = 1
Public Const DPN_SLV_STAT_READY = 2
Public Const DPN_SLV_STAT_READY_DIAG = 3
Public Const DPN_SLV_STAT_NOT_READY = 4
Public Const DPN_SLV_STAT_NOT_READY_DIAG = 5

'*****************************/
'* Defines for masking events */
'*****************************/
Public Const MST_CLS_TWO_ACCESS = 128
Public Const AUTOCLEAR = 64
Public Const WATCHDOG = 32

Public Const DPN_SLV_NO_ACCESS = 0
Public Const DPN_SLV_READ = 1
Public Const DPN_SLV_WRITE_READ = 2
```

These are the error codes that the functions in the dplib.dll library might return.

```
'***************/
'* Error Codes */
'***************/
Public Const DPN_NO_ERROR = 0
Public Const DPN_ACCESS_ERROR = 128
Public Const DPN_APPL_LIMIT_ERROR = 129
Public Const DPN_CENTRAL_ERROR = 130
Public Const DPN_CLOSE_ERROR = 131
Public Const DPN_LENGTH_ERROR = 132
Public Const DPN_MEM_BOARD_ERROR = 133
Public Const DPN_MEM_HOST_ERROR = 134
Public Const DPN_MODE_ERROR = 135
Public Const DPN_NO_DBASE_ERROR = 136
Public Const DPN_OPEN_ERROR = 137
Public Const DPN_RECEIVE_ERROR = 138
Public Const DPN_REFERENCE_ERROR = 139
Public Const DPN_REFERENCE_DIFF_ERROR = 140
Public Const DPN_SEND_ERROR = 141
Public Const DPN_SLV_STATE_ERROR = 142
Public Const DPN_STAT_NR_ERROR = 143
Public Const DPN_USER_DATA_ERROR = 144
Public Const DPN_WRONG_BOARD_ERROR = 145
Public Const DPN_SYS_STATE_ERROR = 146
Public Const DPN_GLB_CTRL_ERROR = 147
Public Const DPN_BOARD_ERROR = 148
Public Const DPN_WD_EXPIRED_ERROR = 149
Public Const DPN_OPEN_LICENSE_ERROR = 150
Public Const DPN_LOAD_L2_VXD_ERROR = 151
Public Const DPN_OPEN_L2_VXD_ERROR = 152
'*********************************************
```

Most of these constants are not used, but are included, as they can be useful when creating new procedures.

### *3.1.1.2 The type declarations*

The functions in the dplib.dll library require a special data structure as parameter. This structure is declared as shown below.

Which parts of this data structure that is in use when making a function call varies from function to function. The `reference_struct` part of it, however, is a key that is always used after initialising the bus. If more applications are running simultaneously using the same controller card, they have different keys identifying them and allowing them access to the card. These keys are called handles. `board_select` is the CP number, which usually is set to 1 when there is only one card present.

```
Type reference_struct
  board_select As Byte
  access As Byte
End Type
```

The other structure elements have the following meaning if they are used:

```
Type dpn_interface
```

This is the handle described above.

```
   reference As reference_struct
```

The slave number.

```
   stat_nr As Byte
```

Number of valid bytes in the `user_data[]` array.

```
   length As Byte
```

Identical to the return parameter of the function.

```
   error_code As Integer
```

Indicates the status of the addressed DP slave.

```
   slv_state As Byte
```

Indicates the current mode of the DP master.

```
sys_state As Byte
```

Identifies any errors occurring on the DP master.

```
sys_event As Byte
```

This array contains the data specific for the job.

```
user_data(MAX_DATA_LEN) As Byte
End Type
```

`board_select` and `stat_nr` are used exclusively as a call value to the dplib.dll functions, whilst `error_code` and `sys_event` are used only as return parameters. The rest of the structure elements are used for both tasks.

### *3.1.1.3 The function headers*

Of the functions in the dplib.dll, the following are declared in this module:

```
Declare Function dpn_init Lib "dplib.dll" (dpn_intf As dpn_interface) As Integer
Declare Function dpn_reset Lib "dplib.dll" (dpn_intf As dpn_interface) As Integer
Declare Function dpn_in_slv Lib "dplib.dll" (dpn_intf As dpn_interface) As Integer
Declare Function dpn_out_slv Lib "dplib.dll" (dpn_intf As dpn_interface) As Integer
Declare Function dpn_read_slv Lib "dplib.dll" (dpn_intf As dpn_interface) As
   Integer
Declare Function dpn_get_mode Lib "dplib.dll" (dpn_intf As dpn_interface) As
   Integer
Declare Function dpn_wd Lib "dplib.dll" (dpn_intf As dpn_interface) As Integer
Declare Function dpn_read_slv_par Lib "dplib.dll" (dpn_intf As dpn_interface) As
   Integer
Declare Function dpn_slv_diag Lib "dplib.dll" (dpn_intf As dpn_interface) As
   Integer
```

Only the five firsts of them are used in the equipment modules in this work. Their descriptions are as follows [19]:

| | |
|---|---|
| dpn_init | This function must be used be the DP application to log on at the CP, and must be called before all other DP functions. It returns a handle needed for later calls by all DP functions. |
| dpn_reset | Deletes the handle of the application and releases the resources assigned to the application. If other applications are logged on, the output data of the slaves assigned to this application are set to 0. If not, the entire DP communication is terminated. |
| dpn_in_slv | Reads the current input data of a DP slave. The data are taken from the local data buffer of the CP, which is updated cyclically. |
| dpn_out_slv | Transfers output data to a DP slave. The output data are entered into the internal memory of the CP. |
| dpn_read_slv | Reads the current output data entered in the local data buffer of the CP. |

## 3.1.2 The procedures module (ProfiProcedures)

All the procedures/functions in this module perform the same actions:

1. They receive data in form of parameters passed to them,
2. put these data into a `dpn_interface` data structure,
3. perform a function call to the dplib.dll library,
4. read data returned in the data structure and
5. return these as a function value.

A detailed description of the various parts of the code in the module follows.

### *3.1.2.1 Declarations*

These are the variables used for keeping values throughout the running of the equipment module.

```
' DP Profibus using the dplib.dll library and a Siemens CP5412 as a master.
'Trond Lieng
'November 1999 - March 2000
'
Option Explicit
```

The `DPAccess` and `DPBoard` variables make up the reference handle that allows access to the library.

```
Public DPAccess As Long
Public DPBoard As Long
```

This variable indicates whether a successful initialisation of the Profibus has been performed or not.

```
Public InitDone As Boolean
```

### *3.1.2.2 Initialise the bus: The InitDP function*

This function initialises the bus and sets the values of the reference handle used in later procedures. It has to be run once when starting the Equipment Module.

*Parameter list and declarations*

This part should explain itself:

```
Public Function InitDP(ByVal BoardNo As Byte, ByVal ReadSlaves, ByVal WriteSlaves)
'Initialises the Profibus
'Both ReadSlaves and WriteSlaves are variant arrays containing the numbers
' of the slaves that are to be initialised with ReadOnly and ReadWrite
' access respectively.
'Trond Lieng Nov 1999
'Continually revised until March 2000
    Dim i As Long, cc As Long
    Dim dpnInterf As dpn_interface
    Dim dpnWatchDog As dpn_interface
    Dim k As Byte
    Dim Address() As Integer
    Dim SlaveNo As Integer
```

`RunTimeError` is a label at the end of the procedure. The procedure goes to this label if an error occurs.

```
On Error GoTo RunTimeError
```

*Checking state of bus and setting the reference parameters*

First, the value of the boolean variable `InitDone` is checked, and the bus is reset if it indicates that the bus is already initialised.

```
'Reset if already initialized
    If InitDone Then
        cc = dpn_reset(dpnInterf)
        If cc <> 0 Then GoTo TheEnd
        InitDone = False
    End If
```

The parts of the data structure that make up the handle are then set.

```
'Choose the number of the CP, then the type and the environment of
'the DP application.
    dpnInterf.reference.board_select = BoardNo
    dpnInterf.reference.access = DPN_SYS_NOT_CENTRAL + DPN_ROLE_NOT_CENTRAL
```

The length of the data structure now indicates the highest slave number. Initially, this is set to zero.

```
dpnInterf.length = 0
```

*Set slave access*

The type of access to each slave has to be set. Initially, the access to all slaves are set to "no access".

```
'Reset all slaves
    For i = 0 To MAX_DATA_LEN
        dpnInterf.user_data(i) = DPN_SLV_NO_ACCESS
    Next
```

Then, all slaves that are to have read-only access are put into the data structure.

```
'Set slaves with Read access only
    For i = 0 To UBound(ReadSlaves)
        SlaveNo = ReadSlaves(i)
        If SlaveNo > 0 And SlaveNo < MAX_SLV_NO Then
            If SlaveNo > dpnInterf.length Then dpnInterf.length = SlaveNo
            dpnInterf.user_data(SlaveNo) = DPN_SLV_READ
        End If
    Next
```

Next, the slaves that are to have write access are set. Any slaves that already are set to have read-only access are overwritten with the new access.

```
'Set slaves with Write/Read access
    '*****************************************************************
    ' WARNING ! Only one application should set read/write access  !!!
    '*****************************************************************
    For i = 0 To UBound(WriteSlaves)
        SlaveNo = WriteSlaves(i)
        If SlaveNo > 0 And SlaveNo < MAX_SLV_NO Then
            If SlaveNo > dpnInterf.length Then dpnInterf.length = SlaveNo
            dpnInterf.user_data(SlaveNo) = DPN_SLV_WRITE_READ
        End If
    Next
```

*Initialising*

First, the `length` part of the data structure has to be checked. It has to have a value one larger than the highest slave number.

```
'Initialise and check for errors
    If dpnInterf.length > 0 Then
        dpnInterf.length = dpnInterf.length + 1
        cc = dpn_init(dpnInterf)
```

If the initialisation was successful, the `InitDone` variable is set to "true" and the handle returned from the `dpn_init` function is saved for later use.

```
If cc = 0 Then
            InitDone = True
            'Return reference handle
            DPBoard = dpnInterf.reference.board_select
            DPAccess = dpnInterf.reference.access
        End If
    End If
```

*The Watchdog*

A watchdog is a small process that checks whether all slaves are still present on the bus and working properly. If it is not the case for any given slave, the output data sent to this slave is set to a "safe value". This option is not presently used, but the code is still included in case one finds a need to use it in the future.

```
'Run without the watchdog, which resets the output in the case of an error.
'If this reset is desired, comment out the next line.
GoTo NoWatchdog
'Set watchdog
    If cc = 0 Then
        dpnWatchDog.reference = dpnInterf.reference
        dpnWatchDog.length = 10
        dpnWatchDog.user_data(0) = 5    '5 * (#400ms) = 2s
        cc = dpn_wd(dpnInterf)
    End If
NoWatchdog:
```

*Ending lines*

Any error code that has occurred during the initialisation process is returned.

```
'Return error code
TheEnd:
InitDP = cc
Exit Function

RunTimeError:
InitDP = Err.Number & ": " & Err.Description
End Function
```

### *3.1.2.3 Write output data: The SendData function*

This function sends an output value to a given channel on a given slave and returns an error code.

*Declarations and check of inputs*

The data that are to be written are sent to the function as a decimal value, together with the slave and channel number.

First, it is checked whether the bus is initialised or not. If it is not, there is something wrong with the use of the module (InitDP not run), or the initialisation failed but the execution of the program did not stop.

The slave and channel number is checked before they are entered into the data structure and are able to cause a run-time error.

*Read current output data*

Because the output is written to all channels in a slave, the current output has to read in order to be able preserve the output of the channels that are not to be changed. These data are then put into the output data structure, together with the access handle and slave number.

*Prepare output data*

The input data in decimal form has to be translated into two hexadecimal bytes of 8 bits each before they are put into the data structure.

The two bytes are put into the `user_data` array as shown. Each channel has two bytes in the array, with the first channel occupying bytes 0 and 1.

*Send output data and return error code*

Finally, all data are sent to the slave, and any error code is returned. The last lines are there to pick up any run-time errors. The line `On Error Goto RunTimeError` in the beginning directs the execution to this point instead of crashing the program.

```
Public Function SendData(ByVal SlaveNo As Integer, ByVal Channel _
                         As Integer, ByVal DecimalData As Long)
'Sends data to an output channel
'Trond Lieng Nov 1999
    Dim i As Long, cc As Long
    Dim dpnSend As dpn_interface
    Dim dpnOldOut As dpn_interface
    Dim Hexdata, Byte1, Byte2

    On Error GoTo RunTimeError

    If InitDone = False Then
        SendData = "Profibus not initialised!"
        Exit Function
    End If

    If SlaveNo < 1 Or Channel < 1 Or SlaveNo > MAX_SLV_NO Or Channel > MAX_CHN_NO
    Then
        SendData = "Slave number or channel number out of range." & Chr(13) _
        & "SlaveNo: " & SlaveNo & ", channel: " & Channel
        Exit Function
    End If

'First read the OUT Buffer of CP
    dpnOldOut.reference.board_select = DPBoard
    dpnOldOut.reference.access = DPAccess
    dpnOldOut.stat_nr = SlaveNo
    dpnOldOut.length = 255
    cc = dpn_read_slv(dpnOldOut)

'Check for errors
    If cc <> 0 Then GoTo ErrorCode
    If dpnOldOut.slv_state <> (DPN_SLV_STAT_READY Or DPN_SLV_STAT_READY_DIAG) Then
        SendData = "Slave " & SlaveNo & " not ready!"
        Exit Function
    End If

'Prepare output data structure
    dpnSend.reference.board_select = DPBoard
    dpnSend.reference.access = DPAccess
    dpnSend.stat_nr = SlaveNo
    dpnSend.length = dpnOldOut.length

'Initially, set all new output bytes equal to the old output bytes
    For i = 0 To dpnOldOut.length - 1
        dpnSend.user_data(i) = dpnOldOut.user_data(i)
    Next

'Turn Decimaldata into two hexadecimal bytes
    Hexdata = Hex$(DecimalData)
    Do While Len(Hexdata) < 4
        Hexdata = "0" & Hexdata
    Loop
    Byte1 = (Format(CLng("&H" & Left(Hexdata, 2))))
    Byte2 = (Format(CLng("&H" & Right(Hexdata, 2))))

'Set new output data for the specified channel
    dpnSend.user_data(2 * Channel - 2) = Byte1
    dpnSend.user_data(2 * Channel - 1) = Byte2

'Send the data of all channels in slave
    cc = dpn_out_slv(dpnSend)

'Return error codes
ErrorCode:
SendData = cc
Exit Function

RunTimeError:
SendData = Err.Number & ": " & Err.Description
End Function
```

This procedure is quite tedious, as it is necessary to send the output data to all channels on a slave instead of only the one in interest. The same procedure for the applicom card is much simpler.

### *3.1.2.4 Read input data: The ReadData function*

This function reads an input value from a given channel on a given slave and returns any error codes.

*Declarations and check of inputs*

The data about the channel that are to be read are sent to the function as `ByVal` parameters. The `DecimalData` parameter is used for returning the input data.

*Prepare data structure and read data*

Again, the data structure has to be prepared before the function call is made.

*Process input data*

As usual with the functions in the dplib.dll library, all input data from the slave are read. The data from the wanted channel must be extracted and transformed into a decimal integer.

*Return error code*

If any error has occurred during the execution of the function, the corresponding error code is returned.

```
Public Function ReadData(ByVal SlaveNo As Byte, ByVal Channel As _
                         Byte, DecimalData As Integer) As Variant
'Reads the data from an input channel.
'Trond Lieng Nov 1999
    Dim dpnReceive As dpn_interface
    Dim cc As Long
    Dim i As Integer
    Dim MaxChannel As Long, offset As Long
    Dim Byte1, Byte2

    On Error GoTo RunTimeError

    If SlaveNo < 1 Or Channel < 1 Or SlaveNo > MAX_SLV_NO Or Channel > MAX_CHN_NO
    Then
        ReadData = "Slave number or channel number out of range." & Chr(13) _
        & "SlaveNo: " & SlaveNo & ", channel: " & Channel
        Exit Function
    End If
    If InitDone Then
        dpnReceive.reference.board_select = DPBoard
        dpnReceive.reference.access = DPAccess
        dpnReceive.stat_nr = SlaveNo
        dpnReceive.length = 255 'This value must be entered here

        cc = dpn_in_slv(dpnReceive)


        If cc = 0 And dpnReceive.slv_state = _
            (DPN_SLV_STAT_READY Or DPN_SLV_STAT_READY_DIAG) Then
            'Number of input channels in the slave
            MaxChannel = Int(dpnReceive.length / 2)
            If MaxChannel < Channel Then
                cc = "Channel number too large!"
                GoTo TheEnd
            End If

            offset = 2 * Channel - 2
            Byte1 = Hex(dpnReceive.user_data(offset))
            'Make sure that 8 bits are entered
                If Len(Byte1) < 2 Then Byte1 = "0" & Byte1
            Byte2 = Hex(dpnReceive.user_data(offset + 1))
                If Len(Byte2) < 2 Then Byte2 = "0" & Byte2
            'Return the decimal value of the input bytes
            DecimalData = CInt("&H" & Byte1 & Byte2)
        Else
            'Check if the slave is the problem
            If dpnReceive.slv_state <> _
             (DPN_SLV_STAT_READY Or DPN_SLV_STAT_READY_DIAG) Then
                ReadData = "Slave " & SlaveNo & " not ready!"
                Exit Function
            End If
        End If
    End If
TheEnd:
ReadData = cc
Exit Function

RunTimeError:
ReadData = Err.Number & ": " & Err.Description
End Function
```

This function is also quite tedious since all input data from the slave is read and the data hence have to be extracted and converted. Again, the Applicom version of this procedure is a little simpler.

### *3.1.2.5 Reset the bus: The ResetDP procedure*

This procedure resets the Profibus, and is usually only executed when terminating the Equipment Module.

This is done by first entering the application's handle into the data stucture and the call the dplib.dll function `dpn_reset`.

```
Public Sub ResetDP()
Dim dpnInterf As dpn_interface, cc As Long
If InitDone Then
    dpnInterf.reference.board_select = DPBoard
    dpnInterf.reference.access = DPAccess
    cc = dpn_reset(dpnInterf)
```

If no errors occurred the `InitDone` variable is set to False, otherwise a message box shows the error number. As the application is terminating, this error message is purely for information purposes. If an error occurred, the CP has to be reset before it can be accessed by a new application.

```
    If cc = 0 Then
        InitDone = False
    Else
        MsgBox "Error code from dpn_reset: " & Hex(cc)
    End If
End If
End Sub
```

#### *3.1.2.6 Other procedures*

There are many possibilities for writing other procedures around the functions of the dplib.dll library. Two of them are included as functions in the ProfiProcedures module:

```
Public Function ReadOutput(ByVal SlaveNo As Byte, ByVal Channel As Byte, _
    OutData As Variant)

Public Function ReadDiag(SlaveNo As Byte)
```

ReadOutput reads the current state of an output channel. It is very similar to ReadData, except that it uses the dplib.dll function `dpn_read_slv` instead of `dpn_in_slv`.

`ReadDiag` returns diagnostics data about a slave. It is based on the dplib.dll function `dpn_slv_diag`. The diagnostics are returned as a variant/string, where the relevant elements of the `user_data[]` return parameter array are put together.

## 3.2 applicom® Profibus

The *applicom*® Profibus [20] implemented here uses an *applicom*® PCI$^{1500PFB}$ card as a master and a library named applicom.dll to control it. The necessary declarations are placed in a module called applicom (Applicom.bas). The module containing the actual procedures is called ProfiProcedures (ProfiApplicom.bas). From a CERN computer, these modules can be found in the folder \\SRV1_ISOLDE\CTL\FecNT\EqpObj\QS_REX\Src (where \\SRV1_ISOLDE\CTL\ is usually mapped as O:\).

### 3.2.1 The declarations module (Applicom)

This module simply contains the headers for the procedure in the applicom.dll library. The library includes almost 150 different procedures including around 40 procedures for writing and 30 procedures for reading data. The ones listed here are the only ones implemented in Visual Basic procedures for the time being.

```
Declare Sub exitbus Lib "applicom" (Stat As Integer)

Declare Sub initbus Lib "applicom" (Stat As Integer)

Declare Sub writepackqbyte Lib "applicom" (Chan As Integer, Nes As Integer, nb As
   Integer, tadr As Long, Tabl As Byte, Stat As Integer)

Declare Sub readpackibyte Lib "applicom" (Chan As Integer, Nes As Integer, nb As
   Integer, tadr As Long, Tabl As Byte, Stat As Integer)

Declare Sub readpackqbyte Lib "applicom" (Chan As Integer, Nes As Integer, nb As
   Integer, tadr As Long, Tabl As Byte, Stat As Integer)
```

#### *3.2.1.1 The Wait Mode and the declared functions*

The read and write functions were chosen in order to provide easy and efficient access to the bus as similar as possible to the functions already used with the Simatic net DP-5412. The three functions chosen are all 'wait mode functions', which means that they provide a synchronous access to the bus.

This type of access works in the way that when a call is made to the bus, the process waits for the answer from the slave before it continues. This is the standard way of accessing the Profibus, and is efficient enough for most purposes although asynchronous procedures exist [21]. The time it takes to make a call and receive the answer is only around 3 ms.

The declared functions have the following descriptions [14]:

| | |
|---|---|
| `initbus` | This function must be executed once in the application program before any access to the other applicom functions. It checks the interface presence and initialises the communication with the applicom library. |
| `exitbus` | This instruction must be executed at the exit of all the tasks that executed an `initbus` (with or without success). Terminates the communication with the applicom library. |
| `writepackqbyte` | Can be used to write output bytes in a device. The different values are stored in the byte table called `'tabl'`. |
| `readpackibyte` | Reads input bytes in a device. The different values are stored in the byte table called `'tabl'`. |
| `readpackqbyte` | Can be used to read output bytes in a device. The different values are stored in the byte table called `'tabl'`. |

The different parameters that the functions above take have the following interpretations:

| | |
|---|---|
| `Chan` | Channel number, which usually is zero. This corresponds to the board number of the applicom card. |
| `Nes` | Equipment number, which is similar to slave number but not necessarily the same as the equipment number can be set manually during the configuration of the Profibus. |
| `nb` | Number of bytes to be read or written. |
| `tadr` | Address of the first byte to be read or written in the device (slave). |
| `Tabl` | Byte table with data that are read or are to be written. Only the first element of this table is transferred as a pointer. |
| `Stat` | Interchange status (error code). |

## 3.2.2 The procedures module (ProfiProcedures)

This module, with the file name ProfiApplicom.bas, contains the same functions as the module with the same name described in section 3.1.2. Where possible the functions have the same name and parameter list in order to maximise transparency when writing Equipment Modules.

However, some differences exist since Applicom and Siemens have chosen different ways to control their Profibus cards. This is most applicable to the InitDP function, where the Applicom version does not have any parameters. Also, the use of the InitDP and ResetDP functions is quite different. With Applicom, initialisation of the bus is done with an own program called PCINIT, which has to be run before the functions in the present module are used. The InitDP and ResetDP functions simply control access to the applicom.dll library, and have to be run before and after the bus is to be accessed from the Equipment Module.

### *3.2.2.1 Declarations*

The declarations' section of the ProfiProcedures (ProfiApplicom.bas) module is as follows:

```
'Procedures for initialising, reading, writing and reseting the
' Applicom DP Profibus.
'Trond Lieng
'February 2000
'
Option Explicit
Option Base 1
```

The Option Base 1 statement means that array indexes starts counting on 1 instead of the default 0. This is important when transferring data to and from the applicom functions with the byte table (see section 3.2.1 above), because the numbering of this table always starts on 1.

```
Public InitDone As Boolean

'The number of the applicom card
Private Const CardNumber = 0
```

The `CardNumber` variable is used in all function calls as the `Chan` parameter (see section 3.2.1 above).

### *3.2.2.2 Establish connection with the applicom.dll library: The InitDP function*

As explained above, this function must be called each time the bus is to be accessed from an Equipment Module. In addition, it allows a check of the interface presence via the return parameter.

The function is very short and simple.
It calls the `initbus` function from the applicom.dll library and sets the `InitDone` variable 'True' if status code zero was returned.
The status code is returned to the procedure calling the `InitDP` function anyway.

```
Public Function InitDP()
Dim cc As Integer

    initbus (cc)
    If cc = 0 Then InitDone = True

InitDP = cc
End Function
```

### *3.2.2.3 Write output data: The SendData function*

This function sends an output value to a given channel on a given slave and returns an error code.

*Declarations*

The data that are to be written are sent to the function as a decimal value, together with the slave and channel number.

```
Public Function SendData(ByVal SlaveNo As Integer, ByVal Channel _
                         As Byte, ByVal DecimalData As Long)
'Sends data to an output channel
'Trond Lieng Feb 2000
    Dim i As Long, cc As Integer
    Dim HexData, Byte1, Byte2
    Dim OutData(2) As Byte
    Dim offset As Long

On Error GoTo RunTimeError
    If InitDone = False Then
        SendData = "Profibus not initialized!"
        Exit Function
    End If
```

First, it is checked whether the bus is initialised or not. If it is not, there is something wrong with the use of the module (InitDP not run), or the initialisation failed but the execution of the program did not stop.

*Prepare output data*

The input data in decimal form has to be translated into two hexadecimal bytes of 8 bits each before they are put into the data table.

```
'Turn Decimaldata into two hexadecimal bytes
    HexData = Hex$(DecimalData)
    Do While Len(HexData) < 4
        HexData = "0" & HexData
    Loop
    Byte1 = (Format(CLng("&H" & Left(HexData, 2))))
    Byte2 = (Format(CLng("&H" & Right(HexData, 2))))

    OutData(1) = Byte1
    OutData(2) = Byte2
```

*Send output data*

The offset is the address of the first byte to be written. Each channel uses two bytes, with the first channel occupying bytes 0 and 1.

```
'First byte to be written
    offset = 2 * Channel - 2
```

The data are then sent to the bus. The number '2' in the parameter list means that two bytes (1 channel) are to be written.

```
'Send the data
    Call writepackqbyte(CardNumber, SlaveNo, 2, offset, OutData(1), cc)
```

*Return error code*

As usual, any error code is returned to the procedure calling the SendData function.

```
'Return error code
SendData = cc
Exit Function

RunTimeError:
SendData = Err.Number & ": " & Err.Description
End Function
```

This function is rather straightforward when compared to the Siemens version in section 3.1.2.3. Counting productive lines of code, the applicom version has only 26 compared to the Siemens version's 44. Considering running times, the figures are about 1.5 ms compared to 8 ms.

### *3.2.2.4 Read input data: The ReadData function*

This function reads an input value from a given channel on a given slave and returns any error codes.

*Declarations and check of inputs*

As with the Siemens version of this function already discussed, the data about the channel that are to be read are sent to the function as `ByVal` parameters. The `DecimalData` parameter is used for returning the input data.

The only check that is done is whether the initialisation is done or not.

```
Public Function ReadData(ByVal SlaveNo As Integer, ByVal Channel As _
                         Byte, DecimalData As Integer) As Variant
'Reads the data from an input channel.
'Trond Lieng Feb 2000
    Dim cc As Integer
    Dim i As Integer
    Dim offset As Long
    Dim Byte1, Byte2
    Dim data(2) As Byte

    On Error GoTo RunTimeError

If InitDone Then
```

*Read and process input data*

Again, the offset is calculated and the applicom.dll procedure is called.
If no error code was returned, the input data are transformed into a decimal integer.

```
    'First byte to be read
    offset = 2 * Channel - 2
    Call readpackibyte(CardNumber, SlaveNo, 2, offset, data(1), cc)

        If cc = 0 Then
            Byte1 = Hex(data(1))
            'Make sure that 8 bits are entered
                If Len(Byte1) < 2 Then Byte1 = "0" & Byte1
            Byte2 = Hex(data(2))
                If Len(Byte2) < 2 Then Byte2 = "0" & Byte2
            'Return the decimal value of the input bytes
            DecimalData = CInt("&H" & Byte1 & Byte2)
```

*Return error code*

An error code is returned. The three different "`ReadData =`" statements below represents the three cases where the `readpackibyte` procedure was called, the bus was not initialised and finally the unlikely case that a run-time error occurred.

```
            ReadData = cc
            Exit Function
        End If
    Else
        ReadData = "Profibus not initialised!"
        Exit Function
    End If

RunTimeError:
ReadData = Err.Number & ": " & Err.Description
End Function
```

### *3.2.2.5 End the communication with the applicom.dll library: The ResetDP function*

This function should be run after all calls to the bus in an Equipment Module procedure is done. It does not reset the bus like the Siemens version does, but simply terminates the connection to the applicom.dll library. Applicom does not appear to have any way of completely resetting the bus.

The function calls the applicom.dll procedure `exitbus`.

The `InitDone` variable is set to 'False', since the InitDP has to be called before next call to the bus whether the `exitbus` procedure went fine or not.

If applicable, an error message is composed and returned. An error from this function usually has no importance, and hence it is not 100 % necessary to use it in the Equipment Module.

```
Public Function ResetDP()
Dim cc As Long
    exitbus (cc)

    InitDone = False

    If cc <> 0 Then ResetDP = "Error code from exitbus: " & cc
End Function
```

### *3.2.2.6 Other procedures*

As with Siemens, there are many possibilities for writing other procedures around the functions of the applicom.dll library. One is included as a function in the ProfiProcedures module:

```
Public Function ReadOutput(ByVal SlaveNo As Integer, ByVal ChannelNo As Byte, _
    OutData As Variant)
```

As its Siemens twin in section 3.1.2.6, it reads the current output state of a given output channel.

# 4 The Equipment Modules

The control system of ISOLDE makes use of so-called Equipment Modules to control the equipment from the front-end computer (FEC). This chapter will first briefly explain how to set up a FEC and write an Equipment Module in Visual Basic. A more complete description of most of the points can be found in "Windows NT as Device Server for the ISOLDE-REX Project" [22]. References to the corresponding chapters in this paper will be given in the text below. The FECs and Equipment Modules made during the work with this thesis will be used as examples.

Secondly, this chapter will include detailed descriptions of two of the Equipment Modules that were made during this work. These are hoped to provide a basic platform for the development of new modules in the future, since this is the weakest point in the existing documentation of the ISOLDE Control System.

The source code of three other modules that were made during this work is included in Appendix B.

## 4.1 The FEC

In order for a computer to be used as a FEC, certain preparations have to be made. Only the most important points will be mentioned here. For more details, see chapter 4.2 of "Windows NT as Device Server for the ISOLDE-REX Project" [22].

First, Windows NT and the needed hardware must be installed. If Profibus is to be used as hardware interface, a Communication Processor card has to be mounted in the PC. See "Installing Profibus on a PC" (Appendix A) for a description of how to do this.

Then the computer must be set up as a FEC in the ISOLDE database. This will be described in more detail below.

Finally, certain modifications must be made in the registry of the computer, the PATH variable, the mapped network drives and the login options.

## 4.2 The Database entries

As already explained in section 2.2.2.1on page 18, all details about the computer and equipment that is to be controlled have to be entered in the ISOLDE database.

The FEC, the Class(es) and all the Equipment need to be registered in the database. How to do this is described in chapter 4.3.1 of "Windows NT as Device Server for the ISOLDE-REX Project" [22].

### 4.2.1 The FEC

For the FEC, the following information must be put into the database:

- Name
- Description
- Hostname
- IP address
- Location
- Operating system
- CPU

Other information is optional.

### 4.2.2 The Class(es)

A FEC can control one or more types of equipment, each defined by a class registered in the database and an ActiveX control (the equipment module). For each class, the following information should be entered into the database:

- Name
- Description
- Property definitions
- Status code definition

Typical properties being used for most classes are shown in the table below.

**Table 4-1: Typical property names and their descriptions. [23]**

| Property name | Description |
|---|---|
| CCV | Current Control Value – the (most important) analogue output to the equipment. |
| AQN | AcQuisitioN - the (most important) analogue input from the Equipment Module. |
| STAQ | Status AcQuisition – status code for the equipment. This is coded in the database, so that each STAQ value represents a status message which is showed to the user in the control panel for the equipment. |
| SCLW | Scaling factor between the CCV value and the output to the DAC. |
| SCLR | Scaling factor between and the input from the ADC and the CCV value. |
| CHNW | Channel number for writing a value to the equipment. |
| CHNR | Channel number for reading a value from the equipment. |
| MAXV | MAXimum Value for the CCV property. |
| MINV | MINimum Value for the CCV property. |

If there are several similar property, such as the analogue output values, they should all begin with the same three letters CCV and then be followed by an identifier like a number (e.g. CCV2 or CCV_V).

### 4.2.3 The Equipment

Each individual piece of equipment must be registered in the database. The following data must be entered:

- Name
- Description
- Equipment type (quadrupoles, scanners, ...)
- Location
- Equipment Module (Class)
- FEC
- Owner (ISOLDE/REX-ISOLDE)
- Equipment number
- Control panel
- Property initialisation values (where applicable)

## *4.3 A general Equipment Module*

When all preparations are made, you are ready to make the specific Equipment Module, which is an ActiveX control made in Visual Basic. A description of how this control has to be built up can be found in chapter 4.3.2 of "Windows NT as Device Server for the ISOLDE-REX Project".

When using Profibus as hardware interface, a typical project structure for the ActiveX can look something like shown in Figure 4-1.

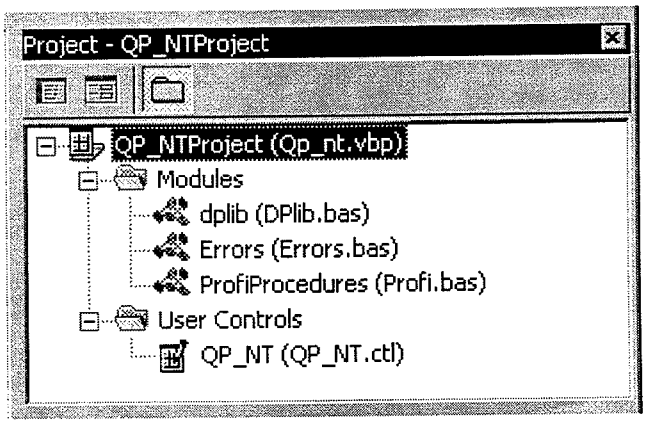



**Figure 4-1: A typical project structure for the Equipment Module**

As seen above, the project consists of one user control and three other modules. The dplib (DPlib.bas) and ProfiProcedures (Profi.bas) are the modules for the Profibus hardware interface and are treated in detail in section 3.1. The Errors (Errors.bas) module simply contains a list of constant declarations of error codes that can be returned from EqpMain/Fecmain.

### 4.3.1 The User Control

The user control contains the connection to EqpMain and hence the control system, and takes care of all equipment calls in the form of property procedures. Some details are described below.

#### *4.3.1.1 Connections to EqpMain*

The connection to EqpMain is made by first adding the EqpMain ActiveX module to the list of components for the Equipment Module project in Visual Basic. Then an instance of it is drawn on the user control object as seen in Figure 4-2 below.

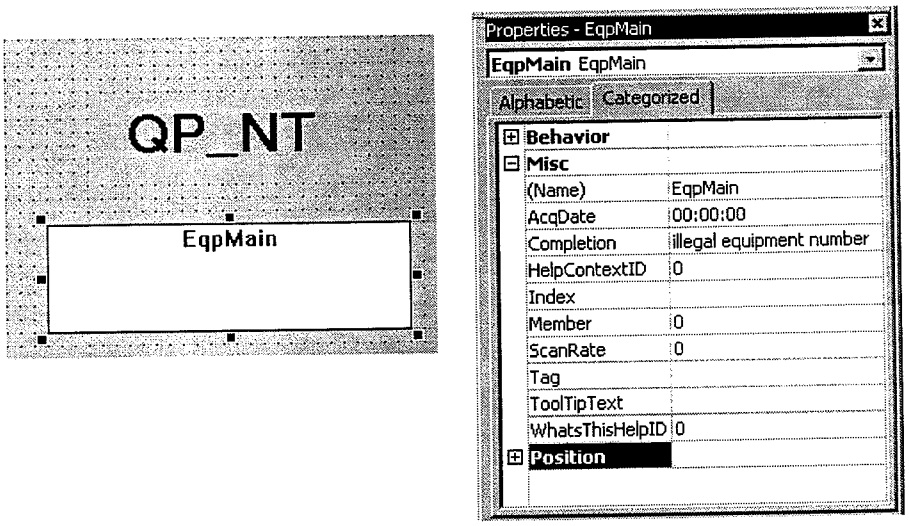



**Figure 4-2: The QP_NT object and some of EqpMain's properties (screenshot from the Visual Basic editor).**

In the declarations' part of the code of the user control, a variable called `eqpMainCtl` is declared:

```
Public eqpMainCtl As Object
```

This variable is being associated with the EqpMain object by including this line in the `UserControl_Initialize()` procedure:

```
Set eqpMainCtl = EqpMain
```

It should be noted that the naming of this variable is not arbitrary, and can not be changed.

Some points should be noted when it comes to the connections to EqpMain:

- All calls to EqpMain (or the object based on it) must be made from the user control, and not from any other modules. (This applies even if the object is declared as public.)
- It is not possible to make calls to the database from the `UserControl_Initialize()` procedure, like `EqpMain.Propertydata` or `EqpMain.AllMemberPropertyData`. This is because the members of the class are initialised after this procedure is run. The initialisation of a hardware interface such as Profibus where property data of the equipment need to be included have to take place later in the start-up sequence of the FEC. In the QP_NT module this is

done by adding a dummy property named INIT that triggers a procedure later on when these database calls are available (see section 4.4.3.4 on page 53).

#### *4.3.1.2 Property Procedures*

Usually, a property requires two procedures, namely "Property Let" and "Property Get". However, in this user control only the "Property Let" method is necessary for R/W properties. For read properties such as AQN and STAQ none of these procedures are used. Instead, the "EqpMain_Callback" event is used.

Only properties that require a special treatment, like equipment access or value checking, need to be implemented in the control. As already explained in previous paragraphs, the 'EQPMAIN' module takes care of the data management of all properties.

In general, a Property Let procedure will contain the following:

- Declaration of variables
- Range check of incoming value
- Insertion of the new value into the EqpMain
- Scaling of the output value
- Reading of addresses and channels for the hardware output
- Writing to hardware
- Return of completion/error code into EqpMain

#### *4.3.1.3 The EqpMain_Callback Event*

This event is fired as often as set in the `EqpMain.ScanRate` property, and is used for reading property values from the equipment and storing them in the data structure. A read request from a user simply retrieves this stored value.

This means that for the user it will not help to try to read these values more frequently than set in the `EqpMain.ScanRate`, for instance by adjusting the `RefreshPeriod` in a control panel [7]. This will only make you read the same data from the data structure several times.

Remember that you will have one call for each member of the class. Make sure that member numbers are stated explicitly when retrieving values from EqpMain.

The property procedures and the code in the `EqpMain_Callback` procedure retrieves all information that is needed from the EqpMain object and the database like slave and channel number and scaling values, before passing these values on to the procedures that take care of the actual hardware calls.

In general, the `EqpMain_Callback` procedure will contain the following for each property to be read:

- Reading of addresses and channels for input data
- Reading of input data from hardware
- Scaling of input data and calculation of equipment property values
- Insertion of property values into EqpMain
- Setting of completion/error codes

#### *4.3.1.4 Other procedures*

It can sometimes be useful to add other procedures in the code of the user control. This should be for instance a standard procedure used by all Property Procedures, as can be seen in the QS_REX Equipment Module (section 4.5.3.7 on page 62).

## 4.4 The QP_NT Equipment Module

This is an Equipment Module made for controlling power supplies connected to quadrupoles in the GPS beamline. Profibus was to be used as the hardware interface, and all programming should be done in Visual Basic. A new control panel for quadrupoles should also be made to replace the existing one made in C++.

From a CERN computer, the source code for this Equipment Module can be found at \\SRV1_ISOLDE\CTL\FecNT\EqpObj\QP_NT\Src\QP_NT.vbp (where \\SRV1_ISOLDE\CTL\ is usually mapped as O:\).

### 4.4.1 Hardware set-up

The task was to control quadrupoles via power supplies of a CERN designed type, called DC24-D3500. These give out 0-3500 V and are controlled by an input voltage of 0-10 V and monitored by a similar output voltage. These voltages were to be transferred by DACs and ACDs connected to a FEC via a Profibus interface.

The Profibus set-up is described in section 2.3.5 on page 25.

The QP_NT module was made on a Siemens-Nixdorf Scenic Pro D5 PC with a *SIMATIC Net DP-5412* Profibus using a CP 5412 card.

In this work, seven power supplies were used. The Simatic slave was set to control the first five power supplies whereas the Wago slave took care of the two last ones. This was done to test difference in performance between the two types of modules.

The DACs and ACDs were connected via a 16-wire flat cable to a specially designed interface board and then to the power supplies with standard Lemo cables.

### 4.4.2 Database Entries

#### *4.4.2.1 The Trondheim FEC*

The computer had to be registered as a FEC. The name "Trondheim" was chosen and put into the database.

#### *4.4.2.2 The QP_NT class*

The class QP_NT had to be created. The names and description of the properties are shown in the table below.

**Table 4-2: Properties in the QP_NT class with a short description and example of initialisation value.**

| Property name | Property description | Example (TRO.QP50) |
|---|---|---|
| ADRR | Read address of the profibus slave | 4 |
| ADRV | Read address of the profibus slave for the vacuum security | 4 |
| ADRW | Write address of the profibus slave | 4 |
| AQN | Read back the acquisition value of the Quadrupole in Volts | |
| CCV | Current control value of the Quadrupole in Volts | |
| CHNR | Input channel in the ADRR AI module | 5 |
| CHNW | Output channel in the ADRW AO module | 5 |
| CHV | Input channel in the ADRV AI module for the Vaccum interlock | 8 |
| INIT | Initialisation of the profibus (dummy property) | 0 |
| MAXOUTPUT | Maximum output value to the DAC | 32511 |
| MAXV | Maximum value of the Quadrupole power supply | 3500 |
| MINV | Minimum value of the Quadrupole power supply | 0 |
| SCLR | Scaling factor for the ADRR AI module | 7.19 |
| SCLV | Scaling factor for the ADRV AI module for the vaccum control | 6912 |
| SCLW | Scaling factor for the ADRW AO module | 7.93 |
| STAQ | Status of the lens power supply | |

All the properties with an initialisation value in the rightmost column are barely fixed constants. The only properties that are to be read or written are CCV, AQN and STAQ.

The coding of the STAQ value was as follows (STAQ value and the corresponding status message):

- 0 = "Invalid"
- 1 = "On"
- 2 = "Off - Vac. Interlock"

#### *4.4.2.3 The equipment*

The seven power supplies were to be connected to one focusing quadropole each, and seven pieces of equipment was registered in the database. The names chosen were TRO.QP10, TRO.QP20, ..., TRO.QP70.

### 4.4.3 The ActiveX Control code

The layout of the ActiveX Control project is shown in Figure 4-1 on page 47, and the layout of the object itself is shown in Figure 4-2 on page 47.

The code of the user control includes the following procedures:

```
Private Sub UserControl_Initialize()

Private Sub EqpMain_Callback(ByVal memNr As Long)
```

```
Public Property Let INIT(ByVal vNewValue As Variant)

Public Property Let CCV(ByVal vNewValue As Variant)

Private Sub UserControl_Terminate()
```

The details of these will be discussed below in the detailed walk-through of the complete code of the QP_NT user control.

### *4.4.3.1 Declarations*

First, the `eqpMaiCtl` is declared as a public object. Later this is set to be an instance of the EqpMain object.

The board number is then specified as a constant. This is the only hardcoded value; all other parameters are in the database

```
'Equipment module: QP_NT
'Written by Trond Lieng
'November 1999
'Last revised April 2000
Option Explicit
Public eqpMainCtl As Object

'The board number of the CP
Private Const DPBoardNumber = 1
```

### *4.4.3.2 The UserControl_Initialize() procedure*

This procedure is run as soon as the AcitveX control is called during the initialisation of the FEC.

This procedure does three simple things:

1. It makes EqpMain accessible via `eqpMainCtl`.
2. It resets the indicator of the Profibus initialisation.
3. It sets the scanrate for the `EqpMain_Callback` event procedure (see below). This is how often the AQN and STAQ properties are refreshed when a hotlink is established (see sections 2.2.2.4 and 4.3.1.3 )

```
Private Sub UserControl_Initialize()


    Set eqpMainCtl = EqpMain


    InitDone = False


'Set scanrate for EqpMain_Callback
    EqpMain.ScanRate = 1500
End Sub
```

### *4.4.3.3 The EqpMain_Callback() procedure*

This procedure reads from the hardware the AQN and STAQ values. It will be called as often as specified in the `EqpMain.Scanrate` property (see above) whenever a hotlink requesting readback values is established.

*Header and declarations*

This procedure is called with one parameter, namely the equipment number. The procedure is called once for each equipment in the class each time the `Callback` event is fired.

```
Private Sub EqpMain_Callback(ByVal memNr As Long)
    Dim AQNInData As Integer, STAQIndata As Integer
    Dim cc As Variant
    Dim Address As Byte
    Dim CHNR As Byte, CHV As Byte
    Dim SCLR As Double, SCLV As Double
    Dim AQN As Long
    Dim STAQ As Long
    Dim STAQValue As Single
```

*The AQN property*

First, the data of the AQN property is read from the Profibus with the `ReadData` function.

```
'Read the input data (AQN)
    Address = EqpMain.PropertyData("ADRR", memNr)
    CHNR = EqpMain.PropertyData("CHNR", memNr)
    cc = ReadData(Address, CHNR, AQNInData)


'Scale the input data
```

The input data is then scaled into the correct voltage value with the SCLR property.

```
    SCLR = EqpMain.PropertyData("SCLR", memNr)
    AQN = CLng(AQNInData / SCLR)
```

The new AQN value is put into the `EqpMain` data structure.
If any error code was returned from `ReadData` this is inserted into the `PropertyCompletion` property.

```
'Insert the new value into EqpMain
    EqpMain.PropertyData("AQN", memNr) = AQN
    If CStr(cc) <> "0" Then EqpMain.PropertyCompletion("AQN", memNr) = cc
```

*The STAQ property*

The STAQ property is a status check of the vacuum. As with the AQN property, the input data is read from the Profibus with the ReadData function.

```
'Read vacuum input (STAQ)
    Address = EqpMain.PropertyData("ADRV", memNr)
    CHV = EqpMain.PropertyData("CHV", memNr)
    'cc = ReadData(Address, CHV, STAQInData)
    cc = ReadData(Address, 2, STAQInData)
```

The scaling is a little different. If the input value is above a threshold value, the vacuum is OK, and the STAQ code is to be '1'.
If the value is below this threshold, there is a problem with the vacuum, and the corresponding STAQ code is '2'.
If an invalid value is received from `ReadData`, the STAQ code is '0'.

The threshold value is given through the SCLV property, and is currently 2.5 V, with a SCLV value of ¼ of the maximum input value of the ADC.

```
'Check vacuum input. Threshold value is given through the SCLV scaling value.
    SCLV = EqpMain.PropertyData("SCLV", memNr)
    STAQValue = CSng(STAQInData / SCLV)
    If (STAQValue > 1) Then
        STAQ = 1    'On

    ElseIf (STAQValue <= 1) Then
        STAQ = 2    'Off - Vac. Interlock

    Else
        STAQ = 0    'Invalid
    End If
```

Finally, the status value is put into the EqpMain data structure together with any error code.

```
'Insert the result into EqpMain
    EqpMain.PropertyData("STAQ", memNr) = STAQ
    If CStr(cc) <> "0" Then EqpMain.PropertyCompletion("STAQ", memNr) = cc
End Sub
```

### *4.4.3.4 The Property Let INIT procedure*

The INIT property is a dummy property that is made just to be able to trigger a procedure late enough in the initialisation process for the information in the database to be available. This is necessary in order to find out which slaves that are to be initialised, since this needs to be specified when using the Simatic net DP-5412 variety of Profibus (see section 3.1.2.2 on page 35).

*Header and declarations*

The `vNewValue` parameter is not used, as this is a dummy property.

```
Public Property Let INIT(ByVal vNewValue As Variant)
'During the initialization procedure of the FEC this procedure is triggered,
'and the Profibus is initialized
    Dim cc, i As Byte
    Dim InSlaves, OutSlaves, VaccuSlaves
```

First it is checked whether the bus is initialised or not. As this procedure is being called once for each equipment in the QP_NT class on the FEC, only the first one will continue any further.

```
'Check if the Profibus is already initialized
    If InitDone = True Then Exit Property

    On Error GoTo ErrorMessage
    Do While cc = 0     'Abort initialization on first error
```

*Read slave numbers from the database*

The `AllMemberPropertyData` method of the EqpMain object returns an array containing all the different values of the specified property of the equipment members of the class registered in the database.

```
'Retrieve the slave numbers of all equipments from the database
    cc = EqpMain.AllMemberPropertyData("ADRW", OutSlaves)
    cc = EqpMain.AllMemberPropertyData("ADRR", InSlaves)
    cc = EqpMain.AllMemberPropertyData("ADRV", VaccuSlaves)
```

It is called three times: for the slaves numbers of the output channels, of the input channels and for the vacuum status channels. The data are returned as an array.

Both the input slave(s) and the vacuum slave(s) are to have read-only access on the Profibus, and are therefore put into one common array.

```
    'Add VaccuSlaves to InSlaves
     ReDim Preserve InSlaves(UBound(InSlaves) + UBound(VaccuSlaves) + 1)
     For i = 0 To UBound(VaccuSlaves)
         InSlaves(UBound(InSlaves) - i) = VaccuSlaves(i)
     Next
```

The InitDP procedure (see section 3.1.2.2) are called, with the two arrays and the DP board number as parameters.

```
'Initialize the Profibus
    cc = InitDP(DPBoardNumber, InSlaves, OutSlaves)
    Exit Do
   Loop
```

Any error code returned from `InitDP` is put into the `EqpMain.Completion` property.

```
'Return error code
    If CStr(cc) <> "0" Then EqpMain.Completion = cc
    Exit Property
```

Run-time errors triggers a message box that will pop up on the screen of the initialising FEC.

```
ErrorMessage:
MsgBox "Error initializing the Profibus!", vbCritical
EqpMain.Completion = "Error initializing the Profibus!"
End Property
```

### *4.4.3.5 The Property Let CCV procedure*

This procedure converts a new CCV value into the corresponding output value of the equipment.

*Header and declarations*

The new CCV value, set by for instance a control panel as in section 0 on page 55 is transferred to this procedure as a Variant.

```
Public Property Let CCV(ByVal vNewValue As Variant)
    Dim CompMsg As String
    Dim OutData As Integer, MaxDecimalOutput As Long
    Dim cc
    Dim Address As Byte
    Dim CHNW As Byte
    Dim SCLW As Double
    Dim CCV As Long
    Dim minv As Double, maxv As Double
```

*Check of the new CCV value*

The new CCV value is converted into an integer (data type *Long*) and checked against the properties MINV and MAXV from the database.

```
'Range check of new CCV value
    CCV = CLng(vNewValue)
    minv = EqpMain.PropertyData("MINV")
    maxv = EqpMain.PropertyData("MAXV")
    If (CCV < minv) Or (CCV > maxv) Then
        EqpMain.Completion = "CCV out of range"
        Exit Property
    End If
```

The CCV value is scaled into the correct output value to the DAC. A check is performed to ensure that this value is not too large so that the output of the DAC becomes zero.

```
'Scale the output data and check range
    SCLW = EqpMain.PropertyData("SCLW")
    MaxDecimalOutput = EqpMain.PropertyData("MAXOUTPUT")
    If (CCV * SCLW) >= MaxDecimalOutput Then
        OutData = MaxDecimalOutput
    Else
        OutData = CInt(CCV * SCLW)
    End If
```

These lines are commented out, as there was currently no need for the extra property CCVD.

```
'The next lines could be used for setting the output value in
'binary (hex) format in the EqpMain
    'CCVDBytes = EqpMain.VariantToByteArray(CCV * SCLW, vbVLong)
    'CCVD = CCVDBytes(0) & CCVDBytes(1)
    'EqpMain.PropertyData("CCVD") = CCVD
```

*Write to hardware*

The scaled output value is sent to the Profibus with the `SendData` function.

```
'Send the new value to the hardware
    Address = EqpMain.PropertyData("ADRW")
    CHNW = EqpMain.PropertyData("CHNW")
    cc = SendData(Address, CHNW, OutData)
```

The error code from SendData, if any, is then put into the `EqpMain.Completion` property.

```
    If CStr(cc) <> "0" Then EqpMain.Completion = cc
```

If the new value was successfully set, the new CCV value is put into the EqpMain data structure.

```
'Insert the new value into EqpMain
    If CStr(cc) = "0" Then EqpMain.PropertyData("CCV") = CCV
```

*Ending lines*

Finally, the `EqpMain_Callback` procedure is called once in order to achieve a near real-time update of the new AQN value.

```
'Readback of AQN and STAQ
Call EqpMain_Callback(ByVal EqpMain.Member)
End Property
```

### *4.4.3.6 The UserContol_Terminate procedure*

This procedure is called if the FecMain window is closed, for instance if the FEC is to be shut down. The Profibus is then reset.

```
Private Sub UserControl_Terminate()
    ResetDP
End Sub
```

On the CERN network, the source code for this Equipment Module can be found in the directory \\SRV1_ISOLDE\CTL\FecNT\EqpObj\QP_NT\Src (where \\SRV1_ISOLDE\CTL\ is usually mapped as O:\).

## 4.4.4 The Control Panel

A new control panel was made for this module. The user interface for this is shown in Figure 4-3.

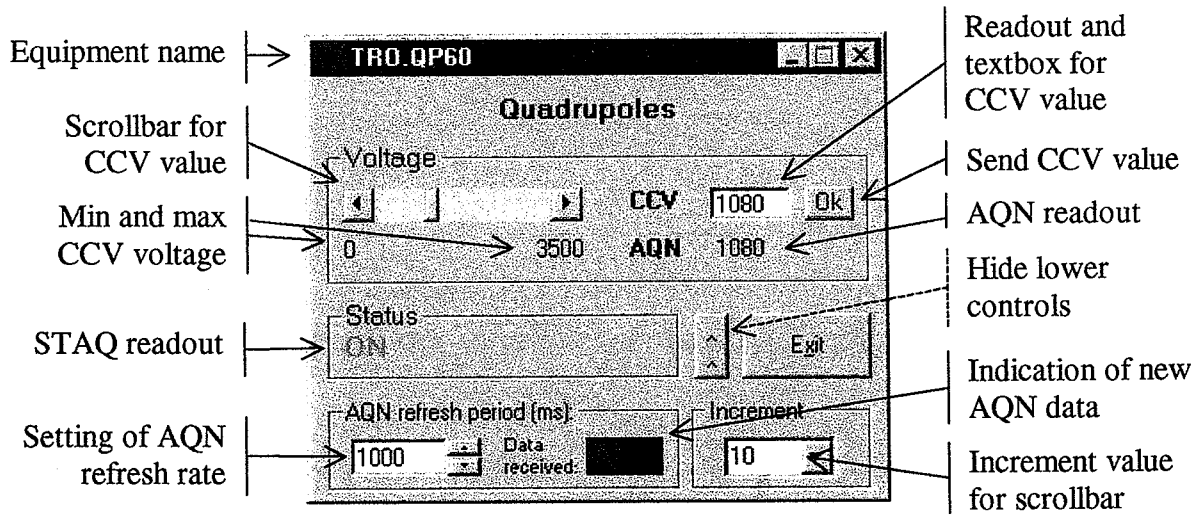



**Figure 4-3: The Control Panel for the QP_NT Equipment Module.**

The layout of the interface is based on an existing control panel for quadrupoles written in C++. It consists exclusively of standard Visual Basic controls.

The connections to the control system are handled through the RpcOle32 object (see section 2.2.2.4).

The source code for this control panel is described in the following, with most of the emphasis being put on the functionality of the control panel and not the use of the RpcOle32 object. The details about this object and how to use it should be found in [7].

*Declarations*

This Boolean determines whether a CCV value is set manually from the text box by clikcing the OK button or by using the scrollbar.

```
'Written by Trond Lieng Feb 2000
Option Explicit
Dim OKBtnPressed As Boolean
```

*Exit*

A standard unloading of the form.

```
Private Sub Exit_Click()
    Unload QPForm
End Sub
```

*Start*

The name of the equipment is taken from the parameter with which the application is called.

All the instances of the RpcOle32 object are initialised to refer to their respective properties of the correct equipment.

Hotlinks (`action = 3`) are created for the AQN, STAQ and CCV properties.

The scrollbar is given the current CCV value.

The interval of the AQN hotlink is entered into the textbox in the lower left corner of the control panel.

```
Private Sub Form_Load()
    QPForm.Caption = Command$
    QPForm.Show

    ' Initialisation of values
    RpcOleMin.ElemName = QPForm.Caption
    RpcOleMin.Property = "MINV"
    RpcOleMin.Action = 1
    RpcOleMax.ElemName = QPForm.Caption
    RpcOleMax.Property = "MAXV"
    RpcOleMax.Action = 1
    RpcOleSTAQ.ElemName = QPForm.Caption
    RpcOleSTAQ.Property = "STAQ"
    RpcOleSTAQ.Action = 3
    'Action 3 is hotlink
    RpcOleAQN.ElemName = QPForm.Caption
    RpcOleAQN.Property = "AQN"
    RpcOleAQN.RefreshPeriod = 1000
    RpcOleAQN.Action = 3
    'RpcOleAQN.Action = 2
    RpcOleCCV.ElemName = QPForm.Caption
    RpcOleCCV.Property = "CCV"
    RpcOleCCV.RefreshPeriod = 3000
    RpcOleCCV.Action = 3
    HScrollVoltage.Value = Val(RpcOleCCV.Text)
    HScrollVoltage.Min = Val(RpcOleMin.Text)
    HScrollVoltage.Max = Val(RpcOleMax.Text)
    MINMAXTab(0) = RpcOleMin.Text
    MINMAXTab(1) = RpcOleMax.Text

    AQNPeriod.Text = RpcOleAQN.RefreshPeriod
End Sub
```

*Using the scrollbar*

This procedure is called when the value of the CCV scrollbar changes. It checks whether the change comes from the OK button being pressed. If it is not, the text of the CCV textbox is set equal to the value of the scrollbar and the new CCV value is sent to the hardware. The hotlink is then re-established in order to detect new CCV values from other users.

```
Private Sub HScrollVoltage_Change()
    If OKBtnPressed = False Then
        RpcOleCCV.Text = HScrollVoltage.Value
        RpcOleCCV.Action = 2
        RpcOleCCV.Action = 3
    End If
End Sub
```

When scrolling the scrollbar, the new CCV value is entered into the text box, but not sent to the hardware. This is done when the scrolling is finished. See above.

```
Private Sub HScrollVoltage_Scroll()
    RpcOleCCV.Text = HScrollVoltage.Value
End Sub
```

*Set a new CCV value*

The value of the CCV property can be set by using the scrollbar as already shown. or by entering a value into the textbox and clicking the OK button. The new value is thoroughly checked before transferred to the scrollbar and the hardware, where an illegal value could do some harm.

```
Private Sub OKVoltage_Click()
Dim NewValue As Integer
    If Val(RpcOleCCV.Text) < 32767 Then
        NewValue = CInt(Val(RpcOleCCV.Text))
    Else
        NewValue = HScrollVoltage.Max
    End If
    If NewValue > HScrollVoltage.Max Then
        NewValue = HScrollVoltage.Max
        RpcOleCCV.Text = NewValue
    ElseIf NewValue < HScrollVoltage.Min Then
        NewValue = HScrollVoltage.Min
        RpcOleCCV.Text = NewValue
    End If
    RpcOleCCV.Action = 2
    RpcOleCCV.Action = 3
    OKBtnPressed = True
    HScrollVoltage.Value = Val(RpcOleCCV.Text)
    OKBtnPressed = False
End Sub
```

*Arrival of new AQN data*

When a new AQN value is received, the label for this value is updated, and the indicator at the bottom of the control panel changes colour.

```
Private Sub RpcOleAQN_NewData()
Dim ColorBuffer As Long
    AQNLabel.Caption = RpcOleAQN.Text
    ColorBuffer = Shape1.FillColor
    Shape1.FillColor = Shape2.FillColor
    Shape2.FillColor = ColorBuffer
End Sub
```

*Arrival of new AQN data*

When an update of the CCV value is received, the value of the scrollbar is updated. This triggers the updating of the textbox as well.

```
Private Sub RpcOleCCV_NewData()
On Error Resume Next
    HScrollVoltage.Value = Val(RpcOleCCV.Text)
End Sub
```

*Arrival of new AQN data*

When a new AQN value is received, the label for this value is updated, with the proper colour (green for ON, red for OFF and blue for INVALID).

```
Private Sub RpcOleSTAQ_NewData()
    LabelSTAQ.Caption = RpcOleSTAQ.Text
    If UCase(RpcOleSTAQ.Text) = "OFF-VAC. INTERLOCK" Then LabelSTAQ.ForeColor =
    &HFF&
    If UCase(RpcOleSTAQ.Text) = "ON" Then LabelSTAQ.ForeColor = &HFF00&
    If UCase(RpcOleSTAQ.Text) = "INVALID" Then LabelSTAQ.ForeColor = &HFF0000
End Sub
```

*Hide or show the lower control*

If the button for hiding or showing the lower controls is pressed, the size of the control panel is changed, and the button is prepared for triggering the opposite operation the next time it is pressed.

```
Private Sub MoreLess_Click()
    If MoreLess.Tag = "more" Then
        Height = Height + 850
        MoreLess.Tag = "less"
        MoreLess.Caption = " ^^"
    Else
        Height = Height - 850
        MoreLess.Tag = "more"
        MoreLess.Caption = "vv"
    End If
End Sub
```

*Change the refresh rate of the AQN value*

If the value in the lower left text box is changed, the `RefreshPeriod` of the AQN hotlink is updated, and the hotlink itself re-established to activate this new value.

```
Private Sub AQNPeriod_Change()
Dim i As Integer
    On Error GoTo TheEnd
    AQNPeriodScroll.Value = Val(AQNPeriod.Text)

    'Update AQN hotlink
        RpcOleAQN.RefreshPeriod = Val(AQNPeriod.Text)
        RpcOleAQN.Action = 3
TheEnd:
End Sub
```

The up and down arrows next to the lower left text box is a compressed vertical scrollbar, that can also be used for adjusting the refresh rate of the AQN value.

```
Private Sub AQNPeriodscroll_Change()
    AQNPeriod.Text = AQNPeriodScroll.Value
End Sub
```

*Change increment value*

The increment value of the CCV scrollbar can be changed with the control in the lower right corner. The value indicates how much the CCV value is changed by a click on the left and right arrows of the scrollbar. A click on the scrollbar itself causes a change of the CCV value ten times as large.

```
Private Sub IncrValue_Change()
Dim i As Byte
Dim NewValue As Long
On Error Resume Next
    NewValue = CLng(Val(IncrValue.Text))
    IncrValue.Text = NewValue
    If NewValue >= 1 And NewValue <= 100 Then
            HScrollVoltage.SmallChange = NewValue
            HScrollVoltage.LargeChange = 10 * NewValue
    ElseIf NewValue < 1 Then
        IncrValue.Text = 1
    ElseIf NewValue > 100 Then
        IncrValue.Text = 100
    End If
End Sub

Private Sub IncrValue_Click()
Call IncrValue_Change
End Sub
```

From a CERN computer, the source code for this control panel can be found at \\SRV1_ISOLDE\CTL\Contrl32\Quadrupole\Src\QP_NTctlpanel.vbp (where \\SRV1_ISOLDE\CTL\ is usually mapped as O:\).

## 4.5 The QS_REX Equipment Module

This module is made for controlling steering quadrupoles in the REX-ISOLDE via an Applicom Profibus. It is not dramatically different from the QP_NT module, but is included since it makes use of the Applicom procedures, and since it has some different properties. The description of these two Equipment Modules together is hoped to give a better overview of how such modules can be made.

The development of this Equipment Module took place on mostly the same equipment as the QP_NT module was made on. The module presented here is therefore in a "raw" version that probably will be slightly modified when it is tested on the hardware it will actually run on when installed in REX-ISOLDE.

From a CERN computer, the source code for this Equipment Module can be found at \\SRV1_ISOLDE\CTL\FecNT\EqpObj\QS_REX\Src\QS_REX.vbp (where \\SRV1_ISOLDE\CTL\ is usually mapped as O:\).

### 4.5.1 Hardware set-up

The FEC used for the development of this Equipment Module was an Olivetti M4 P90I modulo PC with an applicom PCI15000pfb Profibus card.

The rest of the set-up, i.e. the Profibus and the power supplies, was the same as for the QP_NT module. One steering quadrupole requires four different voltages, and the first four of the seven power supplies were used in this work. These were controlled by the first four output and input channels on the Simatic slave.

When installed in REX-ISOLDE, this Equipment Module will actually control a different type of power supplies via Wago and not Simatic I/O modules, but this will only inflict the initialisation values registered in the database.

### 4.5.2 Database Entries

The FEC (named "Haugesund"), the class QS_REX and one piece of equipment (REXTEST.QS10) were registered in the database. The properties are described in the table below.

**Table 4-3: Properties in the QS_REX class with a short description and example of initialisation value.**

| Property name | Property description | Example (REXTEST. QS10) |
|---|---|---|
| ADRR | Profibus slave number for ADCs | 4 |
| ADRV | Profibus slave number for ADCs for Vacuum Interlock | 4 |
| ADRW | Profibus slave number for DACs | 4 |
| AQN | Read Back Voltage | |
| AQN_H | Read Back Horizontal Steering Voltage | |
| AQN_V | Read Back Vertical Steering Voltage | |
| CCV | Set Voltage in Volts | |
| CCV_H | Set Horizontal Steering Voltage. A positive steering voltage turns the beam to the right horizontally and up vertically | |
| CCV_V | Set Vertical Steering Voltage. A positive steering voltage turns the beam to the right horizontally and up vertically | |
| CHNR1 | ADC Channel ( Left Quad Element ) | 1 |
| CHNR2 | ADC Channel ( Right Quad Element ) | 2 |
| CHNR3 | ADC Channel ( Top Quad Element ) | 3 |
| CHNR4 | ADC Channel ( Bottom Quad Element ) | 4 |
| CHNW1 | DAC Channel ( Left Quad Element ) | 1 |
| CHNW2 | DAC Channel ( Right Quad Element ) | 2 |
| CHNW3 | DAC Channel ( Top Quad Element ) | 3 |
| CHNW4 | DAC Channel ( Bottom Quad Element ) | 4 |
| CHV | ADC Channel Vacuum Interlock | 8 |
| MAXLIMIT | Maximum output value to the DAC | 0 |
| MAXV | Max Value of Quad | 32511 |
| MAXV_H | Max Value of Horizontal Steering | 3000 |
| MAXV_V | Max Value of Vertical Steering | 500 |
| MINLIMIT | Minimum output value to the DAC | 500 |
| MINV | Min Value of Quad | 0 |
| MINV_H | Min Value of Horizontal Steering | 500 |
| MINV_V | Min Value of Vertical Steering | -500 |
| SCLR1 | Scaling Factor ADC | -500 |
| SCLR2 | Scaling Factor ADC | 7.15 |
| SCLR3 | Scaling Factor ADC | 7.18 |
| SCLR4 | Scaling Factor ADC | 7.18 |
| SCLV | Scaling Factor ADC Vacuum Interlock | 7.17 |
| SCLW1 | Scaling Factor DAC | 6912 |
| SCLW2 | Scaling Factor DAC | 7.9 |
| SCLW3 | Scaling Factor DAC | 7.92 |
| SCLW4 | Scaling Factor DAC | 7.92 |
| STAQ | Status Acquisition | 7.91 |

## 4.5.3 The ActiveX Control code

The code of this Equipment Module is quite similar to the one of the QP_NT Module. The differences come from the fact that this module uses an applicom Profibus, and that it has three CCV and AQN properties instead of only one.

The two Profibus modules for controlling an applicom Profibus (see section 3.2) were included in the Visual Basic project.

The parts of the code that are very similar to the code of the QP_NT module are not commented in great detail.

### *4.5.3.1 Declarations*

This part has nothing new compared to the QP_NT module.

```
'Equipment module: QS_REX
'Written by Trond Lieng
'March 2000
Option Explicit
Public eqpMainCtl As Object
```

### *4.5.3.2 The UserControl_Initialize() procedure*

Again, this procedure is run as soon as the ActiveX control is called from FecMain. It is identical to the same procedure in the QP_NT module, except that the ScanRate is set to only 400 ms.

```
Private Sub UserControl_Initialize()
    InitDone = False
    Set eqpMainCtl = EqpMain
    EqpMain.ScanRate = 400
End Sub
```

### *4.5.3.3 The EqpMain_Callback() procedure*

This procedure will read three AQN values and the STAQ value from the hardware.

*Header and declarations*

This part has a little more variables than the QP_NT version.

```
Private Sub EqpMain_Callback(ByVal memNr As Long)
    Dim InData1 As Integer, InData2 As Integer, _
        InData3 As Integer, InData4 As Integer
    Dim InVolt1 As Double, InVolt2 As Double, _
        InVolt3 As Double, InVolt4 As Double
    Dim CHNR1 As Byte, CHNR2 As Byte, CHNR3 As Byte, CHNR4 As Byte
    Dim SCLR1 As Double, SCLR2 As Double, SCLR3 As Double, SCLR4 As Double
    Dim STAQIndata As Integer
    Dim cc As Variant
    Dim Address As Byte
    Dim CHV As Byte
    Dim SCLV As Double
    Dim AQN As Long, AQN_H As Long, AQN_V As Long
    Dim STAQ As Long
    Dim STAQValue As Single
```

*Establish connection to the applicom.dll library.*

This part is different from the modules made for the Simatic Net Profibus. The connection to the library has to be re-established every time the module is called upon, since this connection was discovered to be lost between calls. The InitDP function does not really initialise the bus (see section 3.2.2.2).

```
'Initialise the bus
    cc = InitDP
    If cc <> 0 Then
        EqpMain.Completion = "Error initialising the Profibus!"
        Exit Sub
    End If
```

*Read AQN input data*

Since the steering quadrupole

```
'Read the four input values (AQN)
    Address = EqpMain.PropertyData("ADRR", memNr)
    CHNR1 = EqpMain.PropertyData("CHNR1", memNr)
```

uses four output and input channels, all these four values must be read to calculate the three corresponding AQN values.

```
    CHNR2 = EqpMain.PropertyData("CHNR2", memNr)
    CHNR3 = EqpMain.PropertyData("CHNR3", memNr)
    CHNR4 = EqpMain.PropertyData("CHNR4", memNr)
```

The data are read from the input channels, one at a time as long as no error occurs.

```
Do While cc = 0
        cc = ReadData(Address, CHNR1, InData1)
        cc = ReadData(Address, CHNR2, InData2)
        cc = ReadData(Address, CHNR3, InData3)
        cc = ReadData(Address, CHNR4, InData4)
        Exit Do
    Loop

    If CStr(cc) <> "0" Then
        EqpMain.PropertyCompletion("AQN", memNr) = cc
        EqpMain.PropertyCompletion("AQN_H", memNr) = cc
        EqpMain.PropertyCompletion("AQN_V", memNr) = cc
    End If
```

If not all four inputs were successfully read, the error code is returned to the EqpMain. The execution of the procedure continues anyway and tries to calculate the AQN values.

The three AQN values are calculated based on the scaling values registered in the database and the four input values that has just been read. For an explanation of the formulas, see section 4.5.3.7 below on how the CCV values are calculated and section 2.5.2 on the electrode potentials in a steering quadrupole.

```
'Scale the input data
    SCLR1 = EqpMain.PropertyData("SCLR1", memNr)
    SCLR2 = EqpMain.PropertyData("SCLR2", memNr)
    SCLR3 = EqpMain.PropertyData("SCLR3", memNr)
    SCLR4 = EqpMain.PropertyData("SCLR4", memNr)

    InVolt1 = InData1 / SCLR1
    InVolt2 = InData2 / SCLR2
    InVolt3 = InData3 / SCLR3
    InVolt4 = InData4 / SCLR4

    AQN = CLng((InVolt1 + InVolt2 + InVolt3 + InVolt4) / 4#)
    AQN_H = CLng(InVolt3 - InVolt4)
    AQN_V = CLng(InVolt1 - InVolt2)

'Insert the new values into EqpMain
    EqpMain.PropertyData("AQN", memNr) = AQN
    EqpMain.PropertyData("AQN_H", memNr) = AQN_H
    EqpMain.PropertyData("AQN_V", memNr) = AQN_V
```

*The status property (STAQ)*

This is one of the sections of the code that will be modified later, since the desired status codes in REX-ISOLDE has not been decided upon yet.

```
'Read vacuum input (STAQ)
    Address = EqpMain.PropertyData("ADRV", memNr)
    CHV = EqpMain.PropertyData("CHV", memNr)
    cc = ReadData(Address, CHV, STAQIndata)

'Check vacuum input. Threshold value is given through the SCLV scaling value.
    'SCLV = EqpMain.PropertyData("SCLV", memNr)
    'STAQValue = CSng(STAQIndata / SCLV)
    'If (STAQValue > 1) Then
    '    STAQ = 1    'On
    'ElseIf (STAQValue <= 1) Then
    '    STAQ = 2    'Off - Vac. interlock
    'Else
        STAQ = 0    'Invalid
    'End If

'Insert the result into EqpMain
    EqpMain.PropertyData("STAQ", memNr) = STAQ
    If CStr(cc) <> "0" Then EqpMain.PropertyCompletion("STAQ", memNr) = cc
```

*Terminating the connection to the applicom.dll library*

The term 'Reset the bus' is not completely accurate, since this is never done to an applicom Profibus. The ResetDP procedure simply terminates the connestion to the Dynamic Link Library.

```
'Reset the bus
    ResetDP
End Sub
```

### *4.5.3.4 The Property Let CCV procedure*

This procedure is called when a user has set a new value for the CCV property.

```
Public Property Let CCV(ByVal vNewValue As Variant)
    Dim cc
    Dim CCV As Long
    Dim minv As Double, maxv As Double
```

The new CCV value is checked against the MINV and MAXV property values in the database.

If the value was accepted, the new value is entered into the EqpMain data structure.

The SetCCV function (section 4.5.3.7 below) is called in order to calculate the four new output values.

If the setting of the new output values was not successful, the error code is inserted in the EqpMain.

```
'Range check of new CCV value
    CCV = CLng(vNewValue)
    minv = EqpMain.PropertyData("MINV")
    maxv = EqpMain.PropertyData("MAXV")
    If (CCV < minv) Or (CCV > maxv) Then
        EqpMain.Completion = "CCV out of range"
        Exit Property
    End If

'Insert the new value into EqpMain
    EqpMain.PropertyData("CCV") = CCV


'Write all four outputs to hardware
    cc = SetCCV




'Set completion code
    If CStr(cc) <> "0" Then EqpMain.Completion = cc

End Property
```

### *4.5.3.5 The Property Let CCV_H procedure*

This procedure is called when a user has set a new value for the CCV_H property. It is identical to the procedure above, except for the name of the property.

```
Public Property Let CCV_H(ByVal vNewValue As Variant)
    Dim cc
    Dim CCV_H As Long
    Dim minv As Double, maxv As Double

'Range check of new CCV_H value
    CCV_H = CLng(vNewValue)
    minv = EqpMain.PropertyData("MINV_H")
    maxv = EqpMain.PropertyData("MAXV_H")
    If (CCV_H < minv) Or (CCV_H > maxv) Then
        EqpMain.Completion = "CCV_H out of range"
        Exit Property
    End If

'Insert the new value into EqpMain
    EqpMain.PropertyData("CCV_H") = CCV_H

'Write all four outputs to hardware
    cc = SetCCV

'Set completion code
    If CStr(cc) <> "0" Then EqpMain.Completion = cc
End Property
```

### *4.5.3.6 The Property Let CCV_V procedure*

This procedure is called when a user has set a new value for the CCV_V property. It is identical to the two procedures above, except for the name of the property

```
Public Property Let CCV_V(ByVal vNewValue As Variant)
    Dim cc
    Dim CCV_V As Long
    Dim minv As Double, maxv As Double

'Range check of new CCV_V value
    CCV_V = CLng(vNewValue)
    minv = EqpMain.PropertyData("MINV_V")
    maxv = EqpMain.PropertyData("MAXV_V")
    If (CCV_V < minv) Or (CCV_V > maxv) Then
        EqpMain.Completion = "CCV_V out of range"
        Exit Property
    End If

'Insert the new value into EqpMain
    EqpMain.PropertyData("CCV_V") = CCV_V

'Write all four outputs to hardware
    cc = SetCCV

'Set completion code
    If CStr(cc) <> "0" Then EqpMain.Completion = cc
End Property
```

### *4.5.3.7 The SetCCV function*

This function calculates and sets all the four output values based on the three CCV values. It is called from all three Let CCV procedures above.

*Header and declarations*

The function takes no parameters and returns a variant.

```
Private Function SetCCV()
    Dim OutData1 As Integer, OutData2 As Integer, _
        OutData3 As Integer, OutData4 As Integer
    Dim MaxOutput As Long
    Dim CHNW1 As Byte, CHNW2 As Byte, CHNW3 As Byte, CHNW4 As Byte
    Dim SCLW1 As Double, SCLW2 As Double, SCLW3 As Double, SCLW4 As Double
    Dim cc
    Dim Address As Byte
    Dim CCV As Double, CCV_H As Double, CCV_V As Double
```

*Initialisation*

First, the connection to the applicom.dll library must be established.

```
'Initialise the bus
    cc = InitDP
    If cc <> 0 Then
        EqpMain.Completion = "Error initialising the Profibus!"
        Exit Function
    End If
```

*Calculation of the output values*

The four output values are calculated based on the four scaling values form the database and the three CCV values currently set in the `EqpMain` data structure.

```
'Scale the output data
    SCLW1 = EqpMain.PropertyData("SCLW1")
    SCLW2 = EqpMain.PropertyData("SCLW2")
    SCLW3 = EqpMain.PropertyData("SCLW3")
    SCLW4 = EqpMain.PropertyData("SCLW4")

    CCV = EqpMain.PropertyData("CCV")
    CCV_H = EqpMain.PropertyData("CCV_H")
    CCV_V = EqpMain.PropertyData("CCV_V")

    OutData1 = CInt((CCV + CCV_V / 2) * SCLW1)
    OutData2 = CInt((CCV - CCV_V / 2) * SCLW2)
    OutData3 = CInt((CCV + CCV_H / 2) * SCLW3)
    OutData4 = CInt((CCV - CCV_H / 2) * SCLW4)
```

A check is performed to ensure that the calculated output values do not exceed the maximum value the DAC can accept.

```
'Check range of output data
    MaxOutput = EqpMain.PropertyData("MAXLIMIT")
    If OutData1 > MaxOutput Then OutData1 = MaxOutput
    If OutData2 > MaxOutput Then OutData2 = MaxOutput
    If OutData3 > MaxOutput Then OutData3 = MaxOutput
    If OutData4 > MaxOutput Then OutData4 = MaxOutput
```

*Write to hardware*

The four output channels are on the same slave. They are written one by one as long as an error does not occur.

```
'Send new values to hardware
    Address = EqpMain.PropertyData("ADRW")

    CHNW1 = EqpMain.PropertyData("CHNW1")
    CHNW2 = EqpMain.PropertyData("CHNW2")
    CHNW3 = EqpMain.PropertyData("CHNW3")
    CHNW4 = EqpMain.PropertyData("CHNW4")

    Do While cc = 0
        cc = SendData(Address, CHNW1, OutData1)
        cc = SendData(Address, CHNW2, OutData2)
        cc = SendData(Address, CHNW3, OutData3)
        cc = SendData(Address, CHNW4, OutData4)
        Exit Do
    Loop
```

*Ending lines*

The return value of the function is set, the connection to the applicom.dll library is terminated and one readback of AQN and STAQ values are performed to improve the real-time impression of the user.

```
'Return error value
    SetCCV = cc

'Reset the bus
    ResetDP

'Readback of AQN and STAQ
Call EqpMain_Callback(ByVal EqpMain.Member)
End Function
```

One difference from the QP_NT module is that there is no `UserControl_Terminate procedure`. This procedure did only reset the Profibus so that it was ready for new initialisations from new applications. As already explained, this is not necessary when using the applicom Profibus.

The Visual Basic project that make up this Equipment Module also includes three other modules: The Applicom module (section 3.2.1) with the declarations of the functions in the applicom.dll library, the ProfiProcedures module (section 3.2.2) with the Profibus functions, and the Errors module with the FecMain/EqpMain error codes.

On the CERN network, the source code for this Equipment Module can be found in the directory \\SRV1_ISOLDE\CTL\FecNT\EqpObj\QS_REX\Src.

### 4.5.4 The Control Panel

A new control panel was made for this module. The user interface for this is shown in Figure 4-4. The different elements are the same as in the control panel for the QP_NT module, which are explained in Figure 4-3 on page 55. The only difference is that two more sets of controls are added to handle the horizontal and vertical steering voltage, and that it has become a little more compact.

The source code listing for this control panel is included in section B.6.

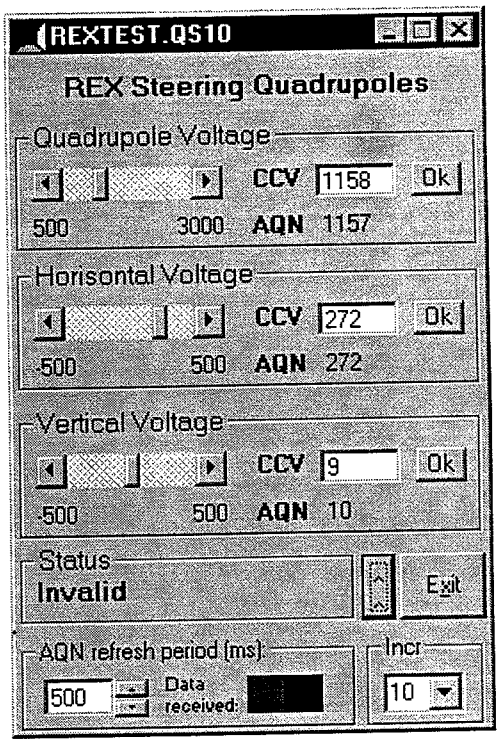



**Figure 4-4: The Control Panel for the QS_NT Equipment Module.**

# 5 Discussion

This chapter will discuss the various subjects that have been treated. The performance of the hardware and software will be discussed, as well as more general remarks on points that will be encountered when installing and using Profibus as well as developing Equipment Modules and control panels.

## 5.1 Programming for the ISOLDE control system

The ISOLDE control system is designed to be as simple as possible when it comes to adding new components such as Equipment Modules and user interfaces. This has been appreciated during this work. Below different aspects of how it is to develop new parts of the control system will be discussed.

### 5.1.1 Programming environment

#### *5.1.1.1 Programming language*

The choice of Visual Basic as programming language is very good in order to make development of new modules as easy as possible. It is not known as the most powerful language, but it shows no signs of weakness for this use. Its user-friendly development environment and good debugging tools has been highly appreciated.

#### *5.1.1.2 EqpTest*

The test container EqpTest, where Equipment Modules can be tested without going through the FecMain application and the actual control system is a handy tool. It clearly simplifies the debugging process since the compiling of the ActiveX control (OCX) is not necessary.

However, one problem was seen. The first Equipment Module developed for the applicom CP seemed to work perfectly in the test container. Later, when compiled and run through the FecMain application, the contact with the Profibus was lost just after initialisation. Eventually the problem was found; the connection to the applicom.dll library was lost between calls to the Equipment Module.

If possible, this should be solved in order to maintain a match as good as possible between the test container and the real environment of the control system.

### 5.1.2 Access to the control system

For most purposes, you only need to use the two "interface objects" EqpMain and RpcOle32 as explained in section 2.2.2 about the software architecture of the control system.

Especially on the user interface side, the RpcOle32 object is very simple and makes the creation of control panels very easy regarding the connection to the control system. The work effort can thereby be put into the functionality of the application.

On the equipment side of the control system, the EqpMain is fairly simple and easy to use. However, the control system has some complex structures that must be understood if problems arise, e.g. the initialisation process of the FEC (EqpInit). It is very easy to add new Equipment Modules as soon at the control system is understood.

The need for being accurate is clearly present. For instance, certain naming rules exist, and the errors that arise if these are broken can come from "deep within" the control system and be quite cryptic like: "FecMain: resources exhausted". As accuracy is something that is always important in programming, this will not be discussed any further, but remain a general statement.

### 5.1.3 General status of the control system and its documentation

The control system is home-made and is still being modified and debugged from time to time. This has certain advantages and certain disadvantages compared to a commercial system.

The advantages are that the system is tailor-made, so that excess parts and functions do not exist. This should provide an easier structure than a commercial system made for a wider market.

Also, the expertise is close, and rapid solutions to problems can usually be expected. This has been appreciated during this work.

However, as mentioned above, the system is still not 100 % debugged. This can cause for quite a lot of frustrations and lost time while searching through potential sources of error. A commercial system has usually been thoroughly debugged.

One important part of the documentation of the control system in its present form is not yet published. This is the PC/CO Note "The REX-ISOLDE Control System Reference" by Ivan Deloose. This paper will probably never be produced, as its author is no longer working with the control system.

All in all, the experience through this work is that the control system is working well, both for daily use and for development and installation of new modules.

## *5.2 Evaluation of the Profibus Hardware*

An important task of this work was to investigate how the different types of Profibus hardware (Communication Processors and Slaves) were functioning. Slave modules from two different vendors and two different CPs were tested. This section will discuss the most important differences, advantages and drawbacks that were found.

### 5.2.1 The Communication Processors

The two Profibus Communication Processors that were tested, are Simatic Net CP 5412 (A2) and *applicom*® PCI[1500PFB]. They were both implemented in the ISOLDE control system without any major difficulties, and both performed without any errors being observed.

The biggest difference between the two cards is the configuration of the Profibus, see Appendix A. The applicom Profibus is much simpler to configure, and even includes an automatic configuration feature. This automatically detects the input and output channels in a slave, so that the user does not have to enter the individual modules in a slave manually. It is even possible to add and remove modules from a slave without having to reconfigure the system.

The programming of the two cards is also a little different, with applicom having both the largest number of available functions and the functions that are easiest to use.

The Simatic CP has one important limitation, which is the access to the slaves. Only one application or Equipment Module can write to the same slave, which means that for instance ordinary quadrupoles and steering quadrupoles must be connected to different slaves. Applicom does not have this limitation.

The applicom card proved to be much simpler and just as reliable to use compared to the more established Simatic card that was already in use at ISOLDE.

### 5.2.2 The slave modules

Both types of modules have performed well during this work. However, the modules are quite different, as can be seen from Table 2-1 on page 26. The following will discuss how these differences inflict the use of the modules.

#### *5.2.2.1 General impression and versatility*

As a general impression, the Simatic slave was very stable and solid in all ways, compared to the smaller and simpler one from Wago, both mechanically and electronically. Then again, neither of them showed any flaws, and the bigger size is more of a drawback when it comes to installation.

*Output ranges*

Concerning versatility, the two slave types are hardly comparable. The Wago modules all operate exclusively in the 0 – 10 V range, whilst the Simatic modules have a multitude of input and output ranges. The output modules' capabilities include six different voltage and current ranges, whereas the input modules are even more multitalented. They can measure voltage, current and resistance, including direct temperature measurements through thermocouples or resistors. In this work, only the 0 –10 V range has been used, so this is more than needed.

*Retaining output value*

There is one feature that is necessary in order to use Profibus for controlling critical components. That is the capability to retain the current output value in case of a failure of the FEC or the Profibus connection. This is important for instance for voltage supplies to target heaters in order to prevent damage of expensive equipment.

This function is built into the Simatic slave, which can be configured in different ways. Using the applicom Profibus, the latest value is always maintained, even if the Profibus cable breaks or the FEC is suddenly shut down. For the Simatic CP5412, the output of each channel can be configured to be the latest output value or any fixed value within the output range.

The Wago buscoupler and output modules do not have this capability.

*Overrange*

The Wago modules have no overrange. They will set the output to zero if a higher decimal value than 32767 is sent to them (overflow). This can happen when outputs close to the maximum is sent to the equipment because the scaling values are set to give the most accurate output over the whole range.

Simatic modules have a substantial overrange, where the decimal value 27648 gives an output of 10 V and the values from 27649 to 32511 give outputs up to about 11.7 V. The output is set to zero only for values over 32511 ($7EFF_H$).

Adding a new property called MAXOUTPUT, containing the value 32767 for the equipment controlled by the Wago modules and 32511 for the equipment controlled by the Siemens modules solved this potential problem. In the Equipment Modules the output value sent to the SendData function is compared to this value and adjusted down to the maximum value if necessary, as shown in sections 3.1.2.3 and 3.2.2.3.

#### *5.2.2.2 Performance*

Both slaves have performed well during this work. After having been calibrated with the correct scaling values, they have given precise and stable output and input values to and from the power supplies.

The largest difference between the slaves, when it comes to performance, is the response time of the input modules. The two different brands use different conversion methods, and the conversion times are thereby quite different as seen in Table 2-1 on page 26. The long conversion time of the Simatic slave makes it difficult to obtain a good real-time feeling, if this is desired.

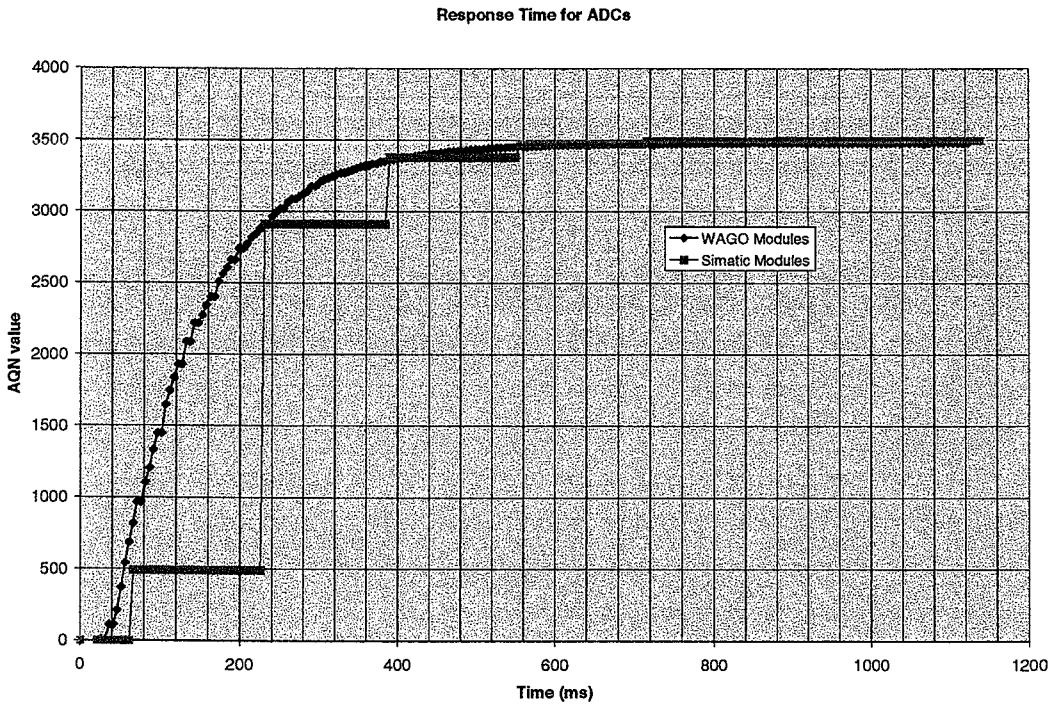



**Figure 5-1: Response of the input modules connected to the power supplies. The CCV value is changed from 0 to 3500 V at t = 0 followed by 200 recordings of the AQN value as quickly as possible.**

Figure 5-1 above shows the response of the two slaves after a sudden change in the CCV value. The test is performed through the test container EqpTest, and shows that the Simatic slave only changes value every 160 ms or so. If a readback is done after 200 ms, the AQN value will be around 500 V for the Simatic slave and 2750 V for the Wago slave.

This has not inflicted the work of this thesis, but is included here as a point to consider for future, more performance-demanding appliances. For instance, the REX-ISOLDE designers want a near-real-time feeling of the control of the quadrupoles, with a minimum of four updates (CCV and AQN) a second. The use of the Simatic modules would have introduced an extra delay of up to 200 ms of the readback signal, which could have weakened the performance. As Wago modules have been chosen as the only type to be used in REX-ISOLDE, this is not a problem.

## *5.3 Performance of DP-Profibus in the ISOLDE control system*

Profibus has already been installed and used in the ISOLDE control system, but the competence on this has left the group. In addition, certain problems has occurred with this installation. It was therefore desired to take a closer look on how Profibus works together with the ISOLDE control system.

### 5.3.1 Adaptability

The Profibus is easy to adapt to the ISOLDE control system. The parameters and data formats are easy for any application to handle. Nearly all the information is structured in a way that the ISOLDE database handles easily. The only exception is the board number, which both the Simatic and the applicom CP makes use of. There is no

natural place to put this into the database. When this is written, the board numbers are hardcoded into the Equipment Modules. When all FECs only have one CP installed, this is no problem since the default number then can be used in all FECs. If a FEC is to have more than one CP installed, the solution must be to add a new property for the board number for all the equipment. The identifying data for one piece of equipment connected to a Profibus will then be:

1. The FEC
2. The board number (s)
3. The slave number(s)
4. The channel number(s)

The only problems observed have been regarding the initialisation of the bus. This has involved both CPs.

The Simatic Net CP5412 requires that only one application has permission to write to a particular slave, whilst all applications can have access to read data from all slaves. "One application" in this regard means one Equipment Module. This implies that an ordinary quadrupole and a steering quadrupole cannot be connected to the same slave using the Simatic Net CP5412. This problem forced the introduction of the dummy property INIT, in order to be able to read property values from the database. With this, it was possible to differentiate which slaves that were to be initialised with which type of access. See section 4.4.3.4 about the `Property Let INIT` procedure.

Also, only four applications are can be connected to the dplib.dll library. This means that a maximum of four different types of equipment can be connected to one FEC using this card. If several cards are installed, the maximum number of four applications in total is still there, and one Equipment Module using both cards will count as two applications. This was not an obstacle in this work, since only two different modules were tested on the FEC with the Simatic card.

The problem with the *applicom*® PCI$^{1500PFB}$ card was that the connection to the applicom.dll library was lost between calls to the Equipment Module. The solution found was to include the InitDP and ResetDP procedure calls in all procedures in the Equipment Modules. This was observed to cause an extra running time of 4.5 – 6.5 ms for the procedures such as the `Property Let CCV` or the `EqpMain_Callback`.

## 5.3.2 Performance

As stated earlier, the DP-Profibus works well in the ISOLDE control system. It is fast and reliable.

### *5.3.2.1 Response time*

Of the time it takes to set a CCV value and read back an AQN value, the time it takes to access the Profibus is only a few percent of the total time, as shown in Figure 5-2 below.

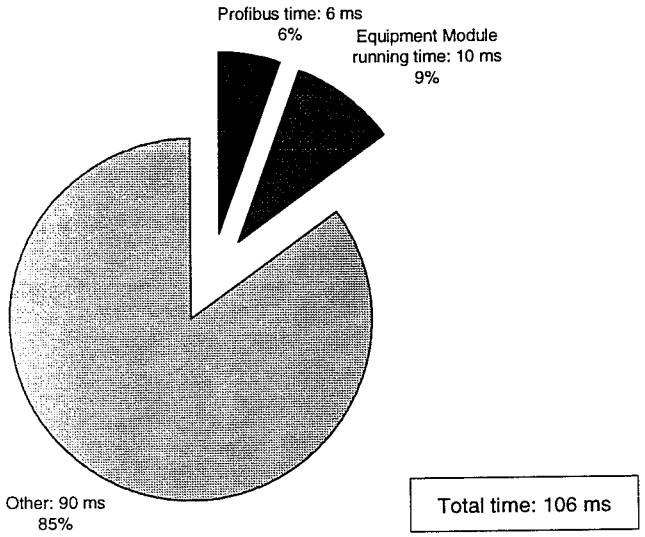



**Figure 5-2: Typical response time for one CCV setting and one AQN readback via a control panel divided into categories. "Other" is network time, processing time in the various software parts of the control system and running time for the control panel.**

The data in this figure was recorded with the 'Info level' in the FecMain application set to '0'. With this level set to the default '2', the extra screen updating on the FEC was observed to take an additional 100 – 150 ms. This probably comes from the fact that the computers used as FECs are usually not very powerful.

The numbers in the figure above is for the QP_NT Equipment Module, which has only one output and input channel for each equipment. More channels will require more calls to the Profibus, which will require and extra 3 ms each. The QS_NT Equipment Module with four output and four input channels was observed to have a typical response time of around 135 ms.

#### *5.3.2.2 Reliability*

In the existing installation of Profibus at ISOLDE, it has been observed that output data has been sent to the wrong channel. This was believed to be caused by the access to the bus or the CP being blocked by an earlier request and that the CP could not handle the new request in a proper way. This should not happen, of course, and it was of some importance to find out whether this was a general problem when using DP-Profibus in the ISOLDE control system.

However, even when running all the available seven pieces of equipment intensively, this was never seen. As long as the calculation of the byte numbers corresponding to the desired channel is programmed correctly, the data seem to arrive at their intended destination. The DP-Profibus and Profibus in general is specifically designed for a high level of transmission security, as explained in section 2.3.2 about the technical specifications of Profibus.

The solution that was found to this problem in the earlier installation of Profibus, was to include a Boolean flag in the Profibus procedures called `DPBusy`. In the beginning of a procedure it was checked whether this flag was 'true', in which case the procedure would wait 100 ms before checking again. When the flag eventually was

found to be 'false', it was "taken" by the procedure and set to 'true' so that other procedure calls would have to wait. This technique was not a good one, and it is hoped that it will still not be found necessary to introduce it again.

### 5.3.3 Full-scale installation

When all this is said about Profibus in the control system, it should be noted that a large-scale installation is yet to be done due to a lack of financial resources. A Profibus installation with several slaves and several dozens pieces of equipment connected to it could worsen the performance.

#### *5.3.3.1 Buffer for input data*

This will especially regard the response time, since the number of calls to the bus could become quite large if numerous pieces of equipment are to run more or less simultaneously. A possible solution has been found, implemented and tested and is ready to be included in the Profibus functions if necessary.

This solution involves the use of a buffer where input data can be stored. This can save some time because all input data from one slave can be read with one call to the bus and put into the buffer. When a new call is made just after the first one, the data will be taken from this buffer and not from the bus or the CP. The buffer includes a time-stamp, and an expiry time is set so that the data being returned from the function is fresh enough.

The implementation of the buffer for the Simatic Net DP-5412 is shown below.

*Declarations*

The following lines are included in the declarations part of the ProfiProcedures module (see section 3.1.2.1 on page 35).

```
Option base 1
```

```
Type BufferType
    LastSlave As Byte
```

`LastSlave` is the number of the slave whose input data are currently in the buffer.

```
    LastRead As Single
```

`LastRead` is the time the data were read from the bus.

```
    Data() As Integer
End Type
```

`Data()` is the array where the input data is stored.

```
Private ReadDataBuffer As BufferType
```

This variable is the buffer, that is kept between calls to the `ReadData` function.

*Use of the buffer*

These are now the first lines in the `ReadData` function (see section 3.1.2.4 on page 38). They are the only ones being executed if the data in the buffer satisfy the demands. The expiry time is here set to two seconds.

```
'First check the buffer, if it is less than 2 secs old and for the correct slave
    If Abs(Timer - ReadDataBuffer.LastRead) < 2# And _
     ReadDataBuffer.LastSlave = SlaveNo Then
        DecimalData = ReadDataBuffer.Data(Channel)
        Exit Function
    End If
```

These lines show how data are being put into the buffer later on in the `ReadData` function.

First, the input data are read from the bus/CP and the time stamp of the buffer is set.

The data array of the buffer is then re-dimensioned to fit the number of channels in the slave.

The data from each of the channels are converted into their decimal value and inserted into the buffer.

The function then returns the correct element of the data array and sets the time stamp.

```
cc = dpn_in_slv(dpnReceive)

    If cc = 0 Then
        'Set time stamp of buffer
        ReadDataBuffer.LastRead = Timer
        'Number of input channels in the slave
        MaxChannel = Int(dpnReceive.length / 2)
        If MaxChannel < Channel Then GoTo Error
        ReDim ReadDataBuffer.Data(MaxChannel)


        For i = 1 To MaxChannel
            Byte1 = Hex(dpnReceive.user_data(2 * Channel - 2))
            'Make sure that 8 bits are entered
            If Len(Byte1) < 2 Then Byte1 = "0" & Byte1
            Byte2 = Hex(dpnReceive.user_data(2 * Channel - 1))
            If Len(Byte2) < 2 Then Byte2 = "0" & Byte2
            'Return the decimal value of the input bytes
            ChannelInput = CInt("&H" & Byte1 & Byte2)
            ReadDataBuffer.Data(i) = ChannelInput
        Next
        'This is what the function returns
        DecimalData = ReadDataBuffer.Data(Channel)

    End If
```

In a currently planned installation of Profibus, up to 30 quadrupoles will be connected to one slave. This buffer technique can then become useful, since the `EqpMain_Callback` event will be fired up to 30 times every 1.5 seconds or so, with each event making at least two calls to the bus (AQN and STAQ value). No change in performance has been seen with the current Profibus test installation with only five pieces of equipment being connected to the same slave.

## 5.4 The Equipment Modules

The Equipment Modules developed in this work are quite simple, with only a few input and output values. They are found to be reliable in the situations they have been tested. They are believed to be ready for a installation in a large scale in the ISOLDE facility.

The modules have been designed to be as simple and clear as possible, in order to serve as examples for the developments of new Equipment Modules for the Profibus interface in the future.

Further development of the Equipment Modules presented in this thesis is still possible, for instance when it comes to more specific error messages. A procedure that converts an error code to an error message could be one idea.

A large-scale installation and consecutive use of the modules will show what type of modifications that are needed and desired.

## 5.5 The user interfaces (control panels)

The user interfaces developed in this work are ready to be taken into use together with their respective Equipment Modules. They have shown a good reliability during the development and testing they have been subject to.

One point should be noted, though: Screen updating seems to take a lot of processor capacity. Very frequent adjustment of CCV values through clicking on scrollbars, especially when several control panels are open at the same time, has caused the stack

to be filled up for a few seconds. A built-in minimum time of 0.2 sec between the acceptance of two consecutive clicks on a scrollbar fixed this problem. The installation and daily use of them will show how the control panels will perform on the quite powerful console PCs in the ISOLDE control room.

It should be noted that the total length of the path and file name of a control panel registered in the ISOLDE database for some reason cannot be longer than 40 characters, or else they will not run when called from the 'Easy equipment access' application.

# 6 Conclusions

Profibus seems to be fast and reliable enough to be used in a large scale at ISOLDE, as already planned. The problems seen in existing installations of Profibus have not been observed during this work.

Both the Simatic and Applicom cards and Simatic and Wago modules are suitable for use with the existing control system. A few differences exist, though. For ordinary uses, with 0 –10 V as input and output range and no special requirements, it is recommended to use an applicom communication processor and Wago slave modules. For more sophisticated use, the Simatic modules are more versatile and should be used e.g. if it is important that the output values are retained in case of failure of the FEC or the Profibus.

The now existing Equipment Modules (Active X controls) work well and are easily converted into new ones.

The control panels made during this work are connected to the Equipment Modules in the database, and are ready to be taken into use when these are installed. New control panels are easily made and maintained with Visual Basic and the RpcOle32 object.

Various techniques for improving the performance of Profibus in the ISOLDE control system and the Equipment Modules have been found and developed, and should be applied in future installations if necessary.

# Appendix A Installing Profibus on a PC

These pages describe step-by-step how to set up Profibus on a PC running on Windows NT. Two different systems are described: *SIMATIC Net DP-5412* using a CP 5412 card and *applicom* using a PCI[1500PFB] card.

## A.1 SIMATIC Net DP-5412

This chapter describes step-by-step how to set up Profibus using Simatic Net DP-5412 with the card CP 5412.

### A.1.1 Hardware Installation

To use Profibus in a PC you have to install a card on the ISA bus. The installation procedure is:

- Choose an address (selected with switches). Each card occupies four continuo addresses in the I/O addresses area starting at the selected base address. The default is 240 (hex).
- Reserve an interrupt in your PC. This can be set by software later on. You can only use interrupt 5, 10, 11, 12 or 15. Default is IRQ 11.
- Put the card into an empty ISA slot in your PC.
- Connect your slaves to the card with a Profibus cable.

### A.1.2 Software and License Installation

Everything described here is described in more detail in the "Installation Instructions CP-5412/Windows NT" that comes with the package from Simatic.

> **Important: You have to log in as a local administrator because the software is going to write to the registry.**

- Insert the first diskette, called SIMATIC NET DP-5412/WINDOWS NT, and run a:\setup.exe. Follow the instructions given by the installation program.
- Then install the diskette called SIMATIC NET COM PROFIBUS WINDOWS (ENGL.).

### A.1.3 Setup and Configuration of the Card and the Profibus

Before the bus is ready for use, one has to specify the selected hardware specifications (address, interrupt and so on) and the equipment connected to the bus.

#### *A.1.3.1 Configuring the bus*

The program COM PROFIBUS enables you to configure the architecture of the bus and the modules inside each slave.

- First choose the Master in **address 1, CP 5412 (A2)** and press OK.

- Then select the **DP configuration**.
- Then you build the architecture of your bus by dragging and dropping elements from the list of Slaves onto your sheet (DP Master system Profibus Address 1). You should first add your **ET 200 Module**, if this is to be used.
- You can now double click on this ET 200 module and press Configure. Select the first vacant row (white cell in first column) by clicking on it, and press **Order no**. This allows you to add your module(s) into the ET 200 Slave. Go now on the same row(s) in the I/O columns (white cells) and press **Auto Address** or enter any other address if desired. The screen could look something like in Figure A-1 below.

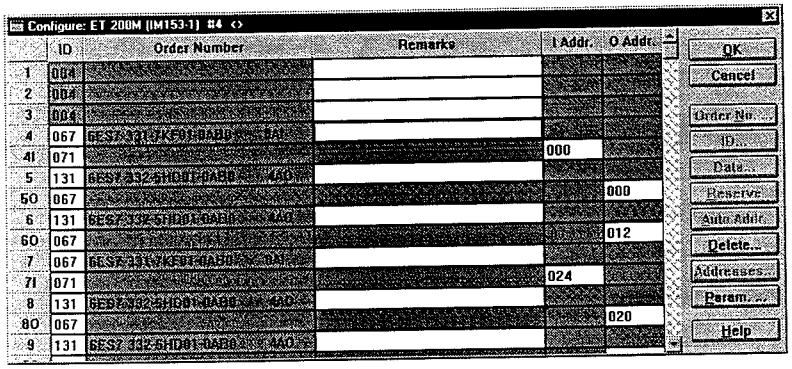



**Figure A-1: Configuring a slave in COM Profibus**

After having added two slaves to your bus, the COM PROFIBUS screen can look something like in Figure A-2.

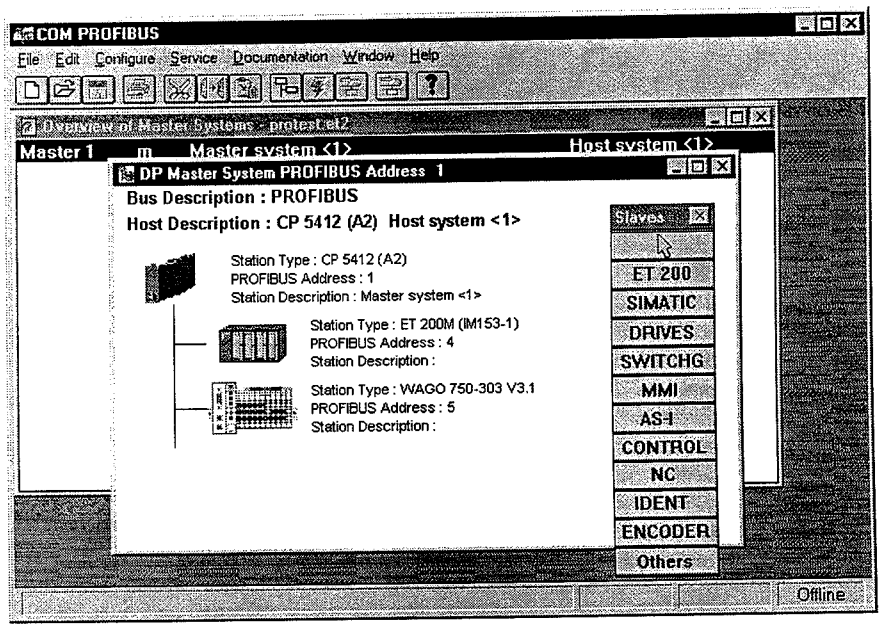



**Figure A-2: A fully configured Profibus**

*Adding a New Slave*

If your slave/buscoupler module is not listed in the COM PROFIBUS program, you will have to copy some files into the appropriate folders on the hard disk. These files describe the new module and the possible input and output modules you can attach to it. They should be found on a diskette enclosed in the package with the module, or

they can be downloaded from the Internet on http://www.profibus.com. The filenames in the brackets are for installing a Wago 750-303 module.

- Copy the appropriate gsd-file (e.g. WAGOB751.gsd) from the diskette to the subdirectory of the folder where the configuration software COM ET 200/WIN or COM PROFIBUS/WIN is installed, e.g. C:\compb30\gsd. Make sure the file has the suffix .gsd and not something like .gse if it is an English version. In the latter case, rename it.
- Copy the type file (e.g. WAB753AX.200) into the subdirectory TYPDAT5X, e.g. C:\compb30\typdat5x.
- Copy the image files (e.g. BUSKOPDN.BMP and BUSKOPDS.BMP) into the subdirectory BITMAPS, e.g. C:\compb30\bitmaps. This makes the program show a picture of the module when you add it to your sheet.
- In the COM PROFIBUS program, select **File, Scan GSD Files**. The module is now available on the list of Slaves under the **Others**-button.

### *A.1.3.2 Activating the Configuration and Setting Up the CP*

The configuration has to be transferred from COM PROFIBUS to the database that is read when the Profibus is initialised.

- First, save the configuration via the **File, Save as...** menu option. Choose any name you want.
- Then, export it by selecting **File, Export, NCM file**. Again, choose any name.

Finally you must specify that this database is to be used when initialising the bus. This is done in different ways depending on the software used. This is how it is done using SINEC Setup:

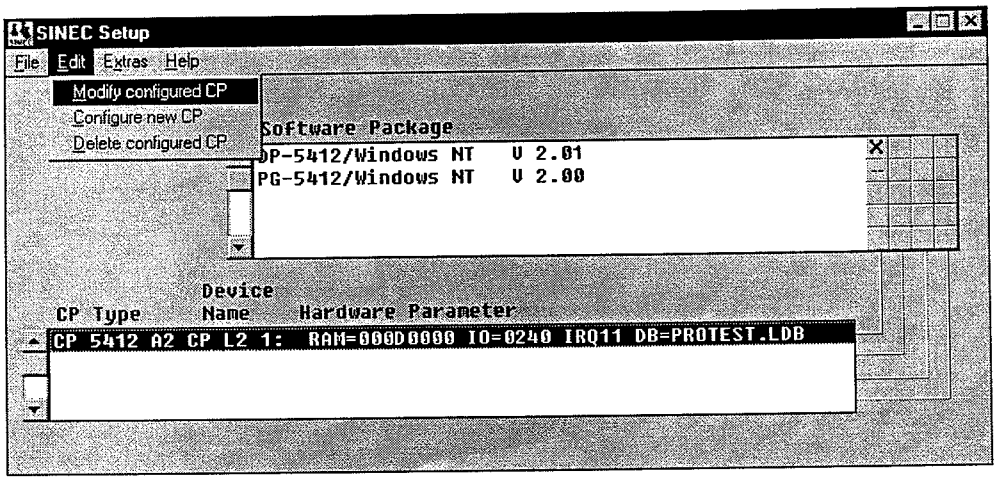



**Figure A-3: Interface of SINEC Setup.**

- Right-click on your card (CP 5412) and select "Modify configured CP" (Figure A-3 above).

- Choose the DP software package and press the **Selection** button to select the database file you just exported.
- In the top frame of the same window you should also enter the correct PC hardware parameters.
- Press OK and restart the computer.

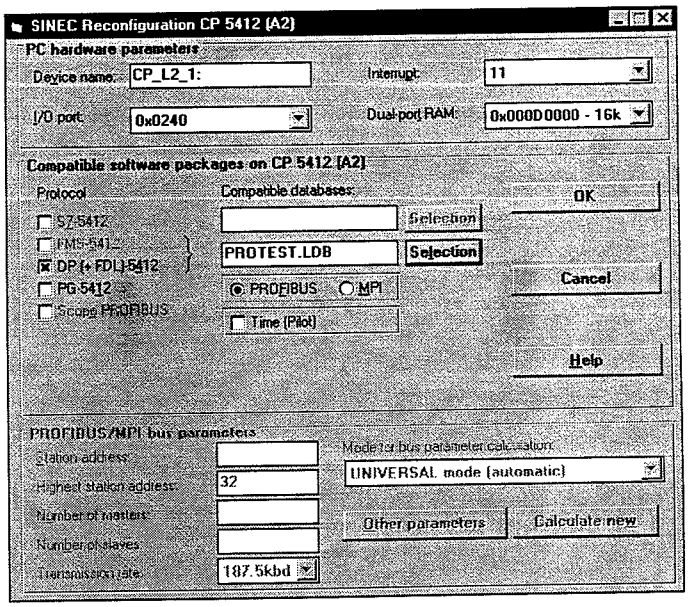



**Figure A-4: Selecting configuration database.**

### A.1.4 Using the test program DPN Demo

The DPN Demo program lets you test the bus you have configured. You have to be logged in as a local administrator to be able to access the registry and thereby the bus. With this program you can run all DP functions, such as initialization of the bus, reading and writing of data, reading diagnostics and so on.

Details regarding the different functions are described in the book "DP Programming Interface".

### A.1.5 Creating an equipment module

Follow chapters 4.2 ("The installation procedure") and 4.3 ("The creation of an equipment control") of "Windows NT as Device Server for the ISOLDE-REX Project" by Ivan Deloose point by point. After this, your FEC is ready and all you have to do is to write the specific code for your equipment.

Sample code can be found in the source code for the existing modules HRSMAGNET and QP_NT.

Unfortunately, no detailed reference manual for writing this code exists. For some more information, see "Creating new FECs and Equipment Modules".

## *A.2 Applicom*

This chapter describes step-by-step how to set up Profibus using the *applicom* PCI$^{1500PFB}$ card.

### A.2.1 Hardware Installation

The installation procedure for the Profibus using the applicom card is very straightforward:

- First, set the three jumpers between the two largest square chips on the card (Figure A-5) so that there is no conflict with other applicom cards in the computer. You can install up to 8 cards in the same computer. If you have no other applicom cards installed, leave the jumpers on the default "000".
- Insert the card into an empty PCI slot in the computer.
- Plug the Profibus cable to the card.

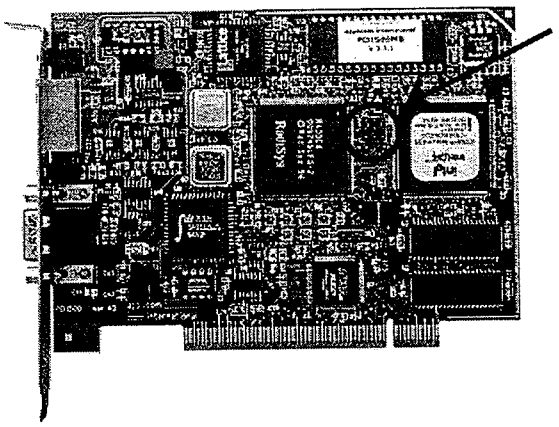


**Figure A-5: Layout of the applicom PCI$^{1500}$ card with the three jumpers.**

## A.2.2 Software Installation

The installation of software is also straightforward:

- Log in on Windows NT as a local administrator.
- Insert the applicom CD-ROM into the drive.
- Wait for the autorun menu or run **setup.exe**.
- Choose language (English or French).
- From the main menu, select **<<Install>>**.
- Select **<<Windows 32-bit NT 4.0 & 95 (Intel)>>**.
- When the window shown in Figure A-6 comes up, select the Profibus protocol.
- Reboot the machine when the installation is completed.

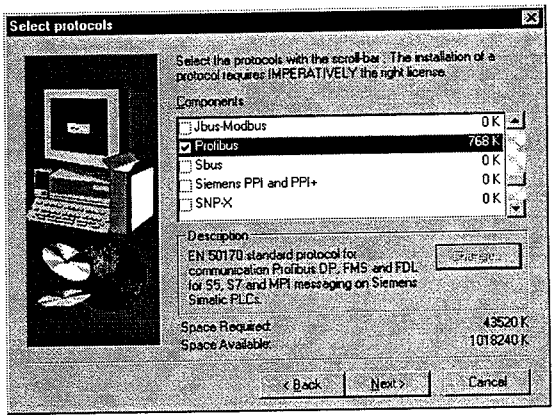



**Figure A-6: Dialog box for selecting protocols.**

## A.2.3 Configuring the Bus

With the applicom card, configuration of the bus is done very easily with a program called PCCONF. The procedure is as follows:

- Choose **Configuration**. Choose card type (PCI1500PFB, check that the PCI radio button is selected). Press **Channels Configuration**. Remember "Highest station address" >= highest slave number.
- Press **Equipments Configuration**, choose any applicom number (not necessarily equal to physical address) and press **Configuration**. In the Messaging list box, select DP PROFIBUS.

- Choose physical address, add a symbolic name and press GSD import. Import the proper gsd file for your slave (gsd files can be downloaded from http://www.profibus.com). After having done this, the dialog box will look something like Figure A-7.
- If you want to use **Automatic I/O configuration**, just press OK. If you want to use **Manual I/O configuration**, choose this option and press **Configuration....** Press **Import GSD** and choose the correct gsd-file. Pick the modules you have and press OK. With automatic configuration you do not have to reconfigure the slave if you change (add/remove) the modules in it later on. You do not even have to reinitialize the bus.
- Configure all slaves in the same way.
- Press OK in all windows until all dialog boxes are closed.
- Choose **File**, **Save and Exit.**
- Reboot and run PCINIT.
- Use Tools (ReadWait, WriteWait) to test the bus.

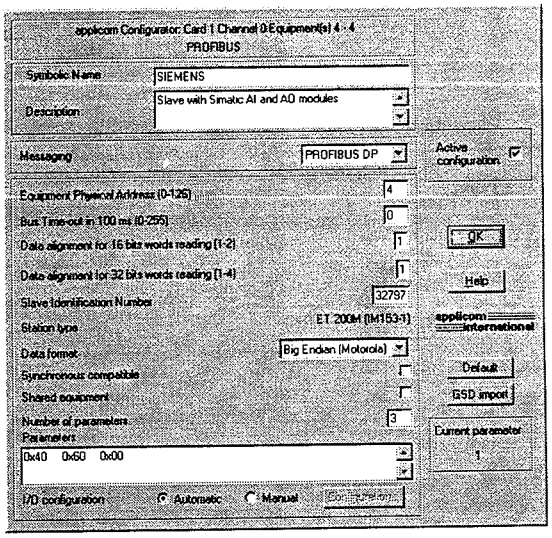



**Figure A-7: Configuring a slave with PCCONF**

# Appendix B Code listing of other modules

This appendix contains the code listing of the user controls, modules and user interfaces that has not been subject of a detailed walk-through previous in this thesis, but that was part of the work.

## B.1 The QP_NTApplicom user control

This Equipment Module is identical in functionality to the QP_NT module, but is made for use with the applicom Profibus.

From a CERN computer, the source code for the Equipment Module that this user control is part of can be found at \\SRV1_ISOLDE\CTL\FecNT\EqpObj\QP_NTApplicom\Src\QP_NTApplicom.vbp (where \\SRV1_ISOLDE\CTL\ is usually mapped as O:\).

```
'Equipment module: QP_NTApplicom
'Written by Trond Lieng
'February 2000
Option Explicit
Public eqpMainCtl As Object

Private Sub UserControl_Initialize()
    InitDone = False
    Set eqpMainCtl = EqpMain
    EqpMain.ScanRate = 1500
End Sub

Private Sub EqpMain_Callback(ByVal memNr As Long)
    Dim AQNInData As Integer, STAQIndata As Integer
    Dim cc As Variant
    Dim Address As Byte
    Dim CHNR As Byte, CHV As Byte
    Dim SCLR As Double, SCLV As Double
    Dim AQN As Long
    Dim STAQ As Long
    Dim STAQValue As Single

'Initialise the bus
    cc = InitDP
    If cc <> 0 Then
        EqpMain.Completion = "Error initialising the Profibus!"
        Exit Sub
    End If

'Read the input data (AQN)
    Address = EqpMain.PropertyData("ADRW", memNr)
    CHNR = EqpMain.PropertyData("CHNR", memNr)
    cc = ReadData(Address, CHNR, AQNInData)

'Scale the input data
    SCLR = EqpMain.PropertyData("SCLR", memNr)
    AQN = CLng(AQNInData / SCLR)

'Insert the new value into EqpMain
    EqpMain.PropertyData("AQN", memNr) = AQN
    If CStr(cc) <> "0" Then EqpMain.PropertyCompletion("AQN", memNr) = cc

'Read vacuum input (STAQ)
    CHV = EqpMain.PropertyData("CHV", memNr)
    'cc = ReadData(Address, CHV, STAQIndata)
    cc = ReadData(Address, 2, STAQIndata)

'Check vacuum input. Threshold value is given through the SCLV scaling value.
    SCLV = EqpMain.PropertyData("SCLV", memNr)
    STAQValue = CSng(STAQIndata / SCLV)
    If (STAQValue > 1) Then
        STAQ = 1    'On
    ElseIf (STAQValue <= 1) Then
        STAQ = 2    'Off - Vac. interlock
    Else
        STAQ = 0    'Invalid
    End If

'Insert the result into EqpMain
    EqpMain.PropertyData("STAQ", memNr) = STAQ
    If CStr(cc) <> "0" Then EqpMain.PropertyCompletion("STAQ", memNr) = cc

'Reset the bus
    ResetDP
End Sub

Public Property Let CCV(ByVal vNewValue As Variant)
```

```
    Dim CompMsg As String
    Dim OutData As Integer, MaxDecimalOutput As Long
    Dim cc
    Dim Address As Byte
    Dim CHNW As Byte
    Dim SCLW As Double
    Dim CCV As Long
    Dim minv As Double, maxv As Double

'Range check of new CCV value
    CCV = CLng(vNewValue)
    minv = EqpMain.PropertyData("MINV")
    maxv = EqpMain.PropertyData("MAXV")
    If (CCV < minv) Or (CCV > maxv) Then
        EqpMain.Completion = "CCV out of range"
        Exit Property
    End If

'Initialise the bus
    cc = InitDP
    If cc <> 0 Then
        EqpMain.Completion = "Error initialising the Profibus! (" & cc & ")"
        Exit Property
    End If

'Scale the output data and check range
    SCLW = EqpMain.PropertyData("SCLW")
    MaxDecimalOutput = EqpMain.PropertyData("MAXOUTPUT")
    If (CCV * SCLW) >= MaxDecimalOutput Then
        OutData = MaxDecimalOutput
    Else
        OutData = CInt(CCV * SCLW)
    End If

'The next lines could be used for setting the output value in
'binary (hex) format in the EqpMain
    'CCVDBytes = EqpMain.VariantToByteArray(CCV * SCLW, vbVLong)
    'CCVD = CCVDBytes(0) & CCVDBytes(1)
    'EqpMain.PropertyData("CCVD") = CCVD

'Send the new value to the hardware
    Address = EqpMain.PropertyData("ADRW")
    CHNW = EqpMain.PropertyData("CHNW")
    cc = SendData(Address, CHNW, OutData)

If CStr(cc) <> "0" Then EqpMain.Completion = cc

'Insert the new value into EqpMain
    If CStr(cc) = "0" Then EqpMain.PropertyData("CCV") = CCV

'Reset the bus
    ResetDP

'Readback of AQN and STAQ
    Call EqpMain_Callback(ByVal EqpMain.Member)
End Property
```

## B.2 The QS_NT user control

This is the main part of an Equipment Module for controlling steering quadrupoles via a Simatic Net CP 5412 version of Profibus. It has three CCV properties named "CCV" for the quadrupole voltage, "CCV2" for the horizontal steering voltage and "CCV3" for the vertical steering voltage. The three corresponding AQN properties are named the same way.

From a CERN computer, the source code for the Equipment Module that this user control is part of can be found at
\\SRV1_ISOLDE\CTL\FecNT\EqpObj\QS_NT\Src\QS_NT.vbp
(where \\SRV1_ISOLDE\CTL\ is usually mapped as O:\).

```
'Equipment module: QS_NT
'Written by Trond Lieng
'January 1999
'
Option Explicit
Public eqpMainCtl As Object

'The board number of the CP
Private Const DPBoardNumber = 1

Private Sub UserControl_Initialize()
    InitDone = False
    Set eqpMainCtl = EqpMain
    EqpMain.ScanRate = 1500
End Sub

Private Sub EqpMain_Callback(ByVal memNr As Long)
    Dim InData1 As Integer, InData2 As Integer, _
        InData3 As Integer, InData4 As Integer
    Dim InVolt1 As Double, InVolt2 As Double, _
        InVolt3 As Double, InVolt4 As Double
    Dim CHNR1 As Byte, CHNR2 As Byte, CHNR3 As Byte, CHNR4 As Byte
    Dim SCLR1 As Double, SCLR2 As Double, SCLR3 As Double, SCLR4 As Double
    Dim STAQIndata As Integer
    Dim cc As Variant
    Dim Address As Byte
    Dim CHV As Byte
    Dim SCLV As Double
    Dim AQN As Long, AQN2 As Long, AQN3 As Long
    Dim STAQ As Long
    Dim STAQValue As Single
```

```
'Read the four input values (AQN)
    Address = EqpMain.PropertyData("ADRR", memNr)
    CHNR1 = EqpMain.PropertyData("CHNR1", memNr)
    CHNR2 = EqpMain.PropertyData("CHNR2", memNr)
    CHNR3 = EqpMain.PropertyData("CHNR3", memNr)
    CHNR4 = EqpMain.PropertyData("CHNR4", memNr)

    Do While cc = 0
    cc = ReadData(Address, CHNR1, InData1)
    cc = ReadData(Address, CHNR2, InData2)
    cc = ReadData(Address, CHNR3, InData3)
    cc = ReadData(Address, CHNR4, InData4)
    Exit Do
    Loop

    If CStr(cc) <> "0" Then
        EqpMain.PropertyCompletion("AQN", memNr) = cc
        EqpMain.PropertyCompletion("AQN2", memNr) = cc
        EqpMain.PropertyCompletion("AQN3", memNr) = cc
    End If

'Scale the input data
    SCLR1 = EqpMain.PropertyData("SCLR1", memNr)
    SCLR2 = EqpMain.PropertyData("SCLR2", memNr)
    SCLR3 = EqpMain.PropertyData("SCLR3", memNr)
    SCLR4 = EqpMain.PropertyData("SCLR4", memNr)

    InVolt1 = InData1 / SCLR1
    InVolt2 = InData2 / SCLR2
    InVolt3 = InData3 / SCLR3
    InVolt4 = InData4 / SCLR4

    AQN = CLng((InVolt1 + InVolt2 + InVolt3 + InVolt4) / 4#)
    AQN2 = CLng(InVolt3 - InVolt4)
    AQN3 = CLng(InVolt1 - InVolt2)

'Insert the new values into EqpMain
    EqpMain.PropertyData("AQN", memNr) = AQN
    EqpMain.PropertyData("AQN2", memNr) = AQN2
    EqpMain.PropertyData("AQN3", memNr) = AQN3

'Read vacuum input (STAQ)
    Address = EqpMain.PropertyData("ADRV", memNr)
    CHV = EqpMain.PropertyData("CHV", memNr)
    'cc = ReadData(Address, CHV, STAQIndata)
    cc = ReadData(Address, 2, STAQIndata)

'Check vacuum input. Threshold value is given through the SCLV scaling value.
    SCLV = EqpMain.PropertyData("SCLV", memNr)
    STAQValue = CSng(STAQIndata / SCLV)
    If (STAQValue > 1) Then
        STAQ = 1    'On
    ElseIf (STAQValue <= 1) Then
        STAQ = 2    'Off - Vac. interlock
    Else
        STAQ = 0    'Invalid
    End If
```

```
'Insert the result into EqpMain
    EqpMain.PropertyData("STAQ", memNr) = STAQ
    If CStr(cc) <> "0" Then EqpMain.PropertyCompletion("STAQ", memNr) = cc

End Sub
Public Property Let INIT(ByVal vNewValue As Variant)
'During the initialization procedure of the FEC this procedure is triggered,
'and the Profibus is initialized
    Dim cc, i As Byte
    Dim InSlaves, OutSlaves, VaccuSlaves

'Check if the Profibus is already initialized
    If InitDone = True Then Exit Property

    On Error GoTo ErrorMessage
   Do While cc = 0      'Abort initialization on first error
'Retrieve the slave numbers of all equipments from the database
    cc = EqpMain.AllMemberPropertyData("ADRW", OutSlaves)
    cc = EqpMain.AllMemberPropertyData("ADRR", InSlaves)
    cc = EqpMain.AllMemberPropertyData("ADRV", VaccuSlaves)

'Add Vaccuslaves to Inslaves
    ReDim Preserve InSlaves(UBound(InSlaves) + UBound(VaccuSlaves) + 1)
    For i = 0 To UBound(VaccuSlaves)
        InSlaves(UBound(InSlaves) - i) = VaccuSlaves(i)
    Next

'Initialize the Profibus
    cc = InitDP(DPBoardNumber, InSlaves, OutSlaves)
   Exit Do
   Loop

'Return error code
    If cc <> 0 Then EqpMain.Completion = cc
    Exit Property

ErrorMessage:
MsgBox "Error initializing the Profibus!", vbCritical
EqpMain.Completion = "Error initializing the Profibus!"
End Property
Public Property Let CCV(ByVal vNewValue As Variant)
    Dim cc As Long
    Dim CCV As Long
    Dim minv As Double, maxv As Double

'Range check of new CCV value
    CCV = CLng(vNewValue)
    minv = EqpMain.PropertyData("MINV")
    maxv = EqpMain.PropertyData("MAXV")
    If (CCV < minv) Or (CCV > maxv) Then
        EqpMain.Completion = "CCV out of range"
        Exit Property
    End If

'Insert the new value into EqpMain
    EqpMain.PropertyData("CCV") = CCV

'Write all four outputs to hardware
    cc = SetCCV
```

```
'Set completion code
    If CStr(cc) <> "0" Then EqpMain.Completion = cc

End Property
Public Property Let CCV2(ByVal vNewValue As Variant)
    Dim cc
    Dim CCV2 As Long
    Dim minv As Double, maxv As Double

'Range check of new CCV2 value
    CCV2 = CLng(vNewValue)
    minv = EqpMain.PropertyData("MINV2")
    maxv = EqpMain.PropertyData("MAXV2")
    If (CCV2 < minv) Or (CCV2 > maxv) Then
        EqpMain.Completion = "CCV2 out of range"
        Exit Property
    End If

'Insert the new value into EqpMain
    EqpMain.PropertyData("CCV2") = CCV2

'Write all four outputs to hardware
    cc = SetCCV

'Set completion code
    If CStr(cc) <> "0" Then EqpMain.Completion = cc
End Property

Public Property Let CCV3(ByVal vNewValue As Variant)
    Dim cc
    Dim CCV3 As Long
    Dim minv As Double, maxv As Double

'Range check of new CCV3 value
    CCV3 = CLng(vNewValue)
    minv = EqpMain.PropertyData("MINV3")
    maxv = EqpMain.PropertyData("MAXV3")
    If (CCV3 < minv) Or (CCV3 > maxv) Then
        EqpMain.Completion = "CCV3 out of range"
        Exit Property
    End If

'Insert the new value into EqpMain
    EqpMain.PropertyData("CCV3") = CCV3

'Write all four outputs to hardware
    cc = SetCCV

'Set completion code
    If CStr(cc) <> "0" Then EqpMain.Completion = cc
End Property

Private Function SetCCV()
    Dim OutData1 As Integer, OutData2 As Integer, _
        OutData3 As Integer, OutData4 As Integer
    Dim MaxOutput As Long
    Dim CHNW1 As Byte, CHNW2 As Byte, CHNW3 As Byte, CHNW4 As Byte
    Dim SCLW1 As Double, SCLW2 As Double, SCLW3 As Double, SCLW4 As Double
    Dim cc
    Dim Address As Byte
    Dim CCV As Double, CCV2 As Double, CCV3 As Double

'Scale the output data
    SCLW1 = EqpMain.PropertyData("SCLW1")
    SCLW2 = EqpMain.PropertyData("SCLW2")
    SCLW3 = EqpMain.PropertyData("SCLW3")
    SCLW4 = EqpMain.PropertyData("SCLW4")

    CCV = EqpMain.PropertyData("CCV")
    CCV2 = EqpMain.PropertyData("CCV2")
    CCV3 = EqpMain.PropertyData("CCV3")

    OutData1 = CInt((CCV + CCV3 / 2) * SCLW1)
    OutData2 = CInt((CCV - CCV3 / 2) * SCLW2)
    OutData3 = CInt((CCV + CCV2 / 2) * SCLW3)
    OutData4 = CInt((CCV - CCV2 / 2) * SCLW4)

'Check range of output data
    MaxOutput = EqpMain.PropertyData("MAXOUTPUT")
    If OutData1 > MaxOutput Then OutData1 = MaxOutput
    If OutData2 > MaxOutput Then OutData2 = MaxOutput
    If OutData3 > MaxOutput Then OutData3 = MaxOutput
    If OutData4 > MaxOutput Then OutData4 = MaxOutput

'Send new values to hardware
    Address = EqpMain.PropertyData("ADRW")

    CHNW1 = EqpMain.PropertyData("CHNW1")
    CHNW2 = EqpMain.PropertyData("CHNW2")
    CHNW3 = EqpMain.PropertyData("CHNW3")
    CHNW4 = EqpMain.PropertyData("CHNW4")

    Do While cc = 0
        cc = SendData(Address, CHNW1, OutData1)
        cc = SendData(Address, CHNW2, OutData2)
        cc = SendData(Address, CHNW3, OutData3)
        cc = SendData(Address, CHNW4, OutData4)
        Exit Do
    Loop

'Return error value
SetCCV = cc

'Readback of AQN and STAQ
Call EqpMain_Callback(ByVal EqpMain.Member)
End Function

Private Sub UserControl_Terminate()
    ResetDP
End Sub
```

## B.3 The QP_REX user control

This is part of the Equipment Module for controlling quadrupoles in the REX-ISOLDE (see section 2.5.3). It has two CCV properties, named "CCV_V" and "CCV_H", and two AQN properties named the same way.

From a CERN computer, the source code for the Equipment Module that this user control is part of can be found at \\SRV1_ISOLDE\CTL\FecNT\EqpObj\QP_REX\Src\QP_REX.vbp (where \\SRV1_ISOLDE\CTL\ is usually mapped as O:\).

```
'Equipment module: QP_REX
'Written by Trond Lieng
'February 2000
Option Explicit
Public eqpMainCtl As Object

Private Sub UserControl_Initialize()
    InitDone = False
    Set eqpMainCtl = EqpMain
    EqpMain.ScanRate = 250
End Sub

Private Sub EqpMain_Callback(ByVal memNr As Long)
    Dim AQNInData_H As Integer, AQNInData_V As Integer, STAQIndata As Integer
    Dim cc As Variant
    Dim Address As Byte
    Dim CHNR_H As Byte, CHNR_V As Byte, CHV As Byte
    Dim SCLR_H As Double, SCLR_V, SCLV As Double
    Dim AQN_H As Long, AQN_V As Long
    Dim STAQ As Long
    Dim STAQValue As Single

'Initialise the bus
    cc = InitDP
    If cc <> 0 Then
        EqpMain.Completion = "Error initialising the Profibus!"
        Exit Sub
    End If

'Read the horisontal and vertical input data (AQN_H and AQN_V)
    Address = EqpMain.PropertyData("ADRR", memNr)
    CHNR_H = EqpMain.PropertyData("CHNR_H", memNr)
    CHNR_V = EqpMain.PropertyData("CHNR_V", memNr)

    cc = ReadData(Address, CHNR_H, AQNInData_H)
    If CStr(cc) <> "0" Then EqpMain.PropertyCompletion("AQN_H", memNr) = cc

    cc = ReadData(Address, CHNR_V, AQNInData_V)
    If CStr(cc) <> "0" Then EqpMain.PropertyCompletion("AQN_V", memNr) = cc

'Scale the input data
    SCLR_H = EqpMain.PropertyData("SCLR_H", memNr)
    SCLR_V = EqpMain.PropertyData("SCLR_V", memNr)
    AQN_H = CLng(AQNInData_H / SCLR_H)
    AQN_V = CLng(AQNInData_V / SCLR_V)

'Insert the new value into EqpMain
    EqpMain.PropertyData("AQN_H", memNr) = AQN_H
    EqpMain.PropertyData("AQN_V", memNr) = AQN_V

'Read vacuum input (STAQ)
    CHV = EqpMain.PropertyData("CHV", memNr)
    'cc = ReadData(Address, CHV, STAQIndata)
    cc = ReadData(Address, 2, STAQIndata)

'Check vacuum input. Threshold value is given through the SCLV scaling value.
    SCLV = EqpMain.PropertyData("SCLV", memNr)
    STAQValue = CSng(STAQIndata / SCLV)
    If (STAQValue > 1) Then
        STAQ = 1    'On
    ElseIf (STAQValue <= 1) Then
        STAQ = 2    'Off - Vac. interlock
    Else
        STAQ = 0    'Invalid
    End If

'Insert the result into EqpMain
    EqpMain.PropertyData("STAQ", memNr) = STAQ
    If CStr(cc) <> "0" Then EqpMain.PropertyCompletion("STAQ", memNr) = cc

'Reset the bus
    ResetDP
End Sub

Public Property Let CCV_H(ByVal vNewValue As Variant)
    Dim OutData As Integer, MaxOutput As Long
    Dim cc
    Dim Address As Byte
    Dim CHNW_H As Byte
    Dim SCLW_H As Double
    Dim CCV_H As Long
    Dim minv_H As Double, maxv_H As Double

'Range check of new CCV_H value
    CCV_H = CLng(vNewValue)
    minv_H = EqpMain.PropertyData("MINV_H")
    maxv_H = EqpMain.PropertyData("MAXV_H")
    If (CCV_H < minv_H) Or (CCV_H > maxv_H) Then
        EqpMain.Completion = "CCV_H out of range"
        Exit Property
    End If

'Initialise the bus
    cc = InitDP
```

```
    If cc <> 0 Then
        EqpMain.Completion = "Error initialising the Profibus!"
        Exit Property
    End If

'Scale the output data and check range
    SCLW_H = EqpMain.PropertyData("SCLW_H")
    MaxOutput = EqpMain.PropertyData("MAXLIMIT")
    If (CCV_H * SCLW_H) >= MaxOutput Then
        OutData = MaxOutput
    Else
        OutData = CInt(CCV_H * SCLW_H)
    End If

'Send the new value to the hardware
    Address = EqpMain.PropertyData("ADRW")
    CHNW_H = EqpMain.PropertyData("CHNW_H")
    cc = SendData(Address, CHNW_H, OutData)

If CStr(cc) <> "0" Then EqpMain.Completion = cc

'Insert the new value into EqpMain
    If CStr(cc) = "0" Then EqpMain.PropertyData("CCV_H") = CCV_H

'Reset the bus
    ResetDP

'Readback of AQN and STAQ
Call EqpMain_Callback(ByVal EqpMain.Member)
End Property

Public Property Let CCV_V(ByVal vNewValue As Variant)
    Dim OutData As Integer, MaxOutput As Long
    Dim cc
    Dim Address As Byte
    Dim CHNW_V As Byte
    Dim SCLW_V As Double
    Dim CCV_V As Long
    Dim minv_V As Double, maxv_V As Double

'Range check of new CCV_V value
    CCV_V = CLng(vNewValue)
    minv_V = EqpMain.PropertyData("MINV_V")
    maxv_V = EqpMain.PropertyData("MAXV_V")
    If (CCV_V < minv_V) Or (CCV_V > maxv_V) Then
        EqpMain.Completion = "CCV_V out of range"
        Exit Property
    End If

'Initialise the bus
    cc = InitDP
    If cc <> 0 Then
        EqpMain.Completion = "Error initialising the Profibus!"
        Exit Property
    End If

'Scale the output data and check range
    SCLW_V = EqpMain.PropertyData("SCLW_V")
    MaxOutput = EqpMain.PropertyData("MAXLIMIT")
    If (CCV_V * SCLW_V) >= MaxOutput Then
        OutData = MaxOutput
    Else
        OutData = CInt(CCV_V * SCLW_V)
    End If

'Send the new value to the hardware
    Address = EqpMain.PropertyData("ADRW")
    CHNW_V = EqpMain.PropertyData("CHNW_V")
    cc = SendData(Address, CHNW_V, OutData)

If CStr(cc) <> "0" Then EqpMain.Completion = cc

'Insert the new value into EqpMain
    If CStr(cc) = "0" Then EqpMain.PropertyData("CCV_V") = CCV_V

'Reset the bus
    ResetDP

'Readback of AQN and STAQ
Call EqpMain_Callback(ByVal EqpMain.Member)
End Property
```

## B.4 The control panel for the QS_NT Equipment Module

From a CERN computer, the source code for this control panel can be found at \\SRV1_ISOLDE\CTL\Contrl32\SteerQuad\Src\QS_NTCtlPanel.vbp (where \\SRV1_ISOLDE\CTL\ is usually mapped as O:\).

```
'Written by Trond Lieng Feb 2000
Option Explicit
Dim OKBtnPressed As Boolean  'Ensures only one CCV call when OK button is pressed
Dim BusyScrolling As Boolean

Private Sub Exit_Click()
    Unload QSForm
End Sub

Private Sub Form_Load()
    Dim i As Integer, Equipment As String
    On Error Resume Next

    QSForm.Caption = Command$
    QSForm.Show
    OKBtnPressed = False

    Equipment = QSForm.Caption
```

```
    'Initialisation of values
    RpcOleMin(1).Property = "MINV"
    RpcOleMax(1).Property = "MAXV"
    RpcOleCCV(1).Property = "CCV"
    RpcOleAQN(1).Property = "AQN"

    RpcOleMin(2).Property = "MINV2"
    RpcOleMax(2).Property = "MAXV2"
    RpcOleCCV(2).Property = "CCV2"
    RpcOleAQN(2).Property = "AQN2"

    RpcOleMin(3).Property = "MINV3"
    RpcOleMax(3).Property = "MAXV3"
    RpcOleCCV(3).Property = "CCV3"
    RpcOleAQN(3).Property = "AQN3"

    For i = 1 To 3
        RpcOleMin(i).ElemName = Equipment
        RpcOleMin(i).Action = 1

        RpcOleMax(i).ElemName = Equipment
        RpcOleMax(i).Action = 1

        RpcOleAQN(i).ElemName = Equipment
        RpcOleAQN(i).RefreshPeriod = 1500
        RpcOleAQN(i).Action = 3
        'RpcOleAQN(i).Action = 1

        RpcOleCCV(i).ElemName = Equipment
        RpcOleCCV(i).Action = 1
        RpcOleCCV(i).RefreshPeriod = 3000
        RpcOleCCV(i).Action = 3

        HScrollVoltage(i).Value = Val(RpcOleCCV(i).Text)

        HScrollVoltage(i).Min = Val(RpcOleMin(i).Text)
        HScrollVoltage(i).Max = Val(RpcOleMax(i).Text)

        MinValue(i).Caption = RpcOleMin(i).Text
        MaxValue(i).Caption = RpcOleMax(i).Text
    Next

    RpcOleSTAQ.ElemName = Equipment
    RpcOleSTAQ.Property = "STAQ"
    RpcOleSTAQ.RefreshPeriod = 2000
    RpcOleSTAQ.Action = 3

    AQNPeriod.Text = RpcOleAQN(1).RefreshPeriod
End Sub

Private Sub HScrollVoltage_Change(Index As Integer)
    If OKBtnPressed = False Then
        RpcOleCCV(Index).Text = HScrollVoltage(Index).Value
        'Next lines prevent filling the stack with waiting clicks
        'The BusyScrolling flag is reset by Timer1
        If BusyScrolling Then Exit Sub
        BusyScrolling = True
        RpcOleCCV(Index).Action = 2
        RpcOleCCV(Index).Action = 3
    End If
    OKBtnPressed = False
End Sub

Private Sub HScrollVoltage_Scroll(Index As Integer)
    RpcOleCCV(Index).Action = 0
    RpcOleCCV(Index).Text = HScrollVoltage(Index).Value
End Sub

Private Sub OKVoltage_Click(Index As Integer)
On Error Resume Next
Dim NewValue As Integer
    If Val(RpcOleCCV(Index).Text) < 32767 Then
        NewValue = CInt(Val(RpcOleCCV(Index).Text))
    Else
        NewValue = HScrollVoltage(Index).Max
    End If
    If NewValue > HScrollVoltage(Index).Max Then
        NewValue = HScrollVoltage(Index).Max
        RpcOleCCV(Index).Text = NewValue
    ElseIf NewValue < HScrollVoltage(Index).Min Then
        NewValue = HScrollVoltage(Index).Min
        RpcOleCCV(Index).Text = NewValue
    End If
    RpcOleCCV(Index).Action = 2
    RpcOleCCV(Index).Action = 3
    OKBtnPressed = True
    HScrollVoltage(Index).Value = NewValue
End Sub

Private Sub RpcOleCCV_NewData(Index As Integer)
On Error Resume Next
    If Not BusyScrolling Then HScrollVoltage(Index).Value =
     CInt(Val(RpcOleCCV(Index).Text))
End Sub

Private Sub RpcOleSTAQ_NewData()
On Error Resume Next
    LabelSTAQ.Caption = RpcOleSTAQ.Text
    If UCase(RpcOleSTAQ.Text) = "OFF-VAC. INTERLOCK" Then LabelSTAQ.ForeColor = &HFF&
    If UCase(RpcOleSTAQ.Text) = "ON" Then LabelSTAQ.ForeColor = &HFF00&
    If UCase(RpcOleSTAQ.Text) = "INVALID" Then LabelSTAQ.ForeColor = &HFF0000
End Sub

Private Sub RpcOleAQN_NewData(Index As Integer)
Dim ColorBuffer As Long
On Error Resume Next
    AQNLabel(Index).Caption = RpcOleAQN(Index).Text
    ColorBuffer = Shape1.FillColor
    Shape1.FillColor = Shape2.FillColor
    Shape2.FillColor = ColorBuffer
End Sub

Private Sub Timer1_Timer()
    BusyScrolling = False
End Sub

Private Sub MoreLess_Click()
On Error Resume Next
```

```
    If MoreLess.Tag = "more" Then
        Height = Height + 900
        MoreLess.Tag = "less"
        MoreLess.Caption = " ^^"
    Else
        Height = Height - 900
        MoreLess.Tag = "more"
        MoreLess.Caption = "vv"
    End If
End Sub

Private Sub AQNPeriod_Change()
Dim i As Integer
    On Error GoTo TheEnd
    AQNPeriodScroll.Value = Val(AQNPeriod.Text)

    'Update AQN hotlinks
    For i = 1 To 3
        RpcOleAQN(i).RefreshPeriod = Val(AQNPeriod.Text)
        RpcOleAQN(i).Action = 3
    Next
TheEnd:
Err = 0
End Sub

Private Sub AQNPeriodscroll_Change()
On Error Resume Next
    AQNPeriod.Text = AQNPeriodScroll.Value
End Sub

Private Sub IncrValue_Change()
Dim i As Byte
Dim NewValue As Long
On Error Resume Next
    NewValue = CLng(Val(IncrValue.Text))
    IncrValue.Text = NewValue
    If NewValue >= 1 And NewValue <= 100 Then
        For i = 1 To 3
            HScrollVoltage(i).SmallChange = NewValue
            HScrollVoltage(i).LargeChange = 10 * NewValue
        Next
    ElseIf NewValue < 1 Then
        IncrValue.Text = 1
    ElseIf NewValue > 100 Then
        IncrValue.Text = 100
    End If
End Sub

Private Sub IncrValue_Click()
Call IncrValue_Change
End Sub
```

## B.5 The control panel for the QP_REX Equipment Module

From a CERN computer, the source code for this control panel can be found at \\SRV1_ISOLDE\CTL\Contrl32\REX_Quad\Src\QP_REXctlpanel.vbp (where \\SRV1_ISOLDE\CTL\ is usually mapped as O:\).

```
'Written by Trond Lieng
'February 2000
Option Explicit
Dim OKBtnPressed As Boolean
Dim OneVoltage As Boolean
Dim ActiveProperty As Byte

Private Sub chkOneVoltage_Click()
    OneVoltage = CBool(chkOneVoltage.Value)
End Sub

Private Sub Exit_Click()
    Unload QP_REXForm
End Sub

Private Sub Form_Load()
    Dim i As Integer, Equipment As String
    On Error Resume Next

    QP_REXForm.Caption = Command$
    QP_REXForm.Show
    Equipment = QP_REXForm.Caption

    'Initialisation of values
    RpcOleMin(0).Property = "MINV_H"
    RpcOleMax(0).Property = "MAXV_H"
    RpcOleAQN(0).Property = "AQN_H"
    RpcOleCCV(0).Property = "CCV_H"

    RpcOleMin(1).Property = "MINV_V"
    RpcOleMax(1).Property = "MAXV_V"
    RpcOleAQN(1).Property = "AQN_V"
    RpcOleCCV(1).Property = "CCV_V"

    For i = 0 To 1
        RpcOleMin(i).ElemName = Equipment
        RpcOleMin(i).Action = 1
        RpcOleMax(i).ElemName = Equipment
        RpcOleMax(i).Action = 1
        'Action 3 is hotlink, 1 is read, 2 is write
        RpcOleAQN(i).ElemName = Equipment
        RpcOleAQN(i).Action = 1
        RpcOleCCV(i).ElemName = Equipment
```

```
        RpcOleCCV(i).RefreshPeriod = 3000
        RpcOleCCV(i).Action = 1
        RpcOleCCV(i).Action = 3

        HScrollVoltage(i).Value = Val(RpcOleCCV(i).Text)
        HScrollVoltage(i).Min = Val(RpcOleMin(i).Text)
        HScrollVoltage(i).Max = Val(RpcOleMax(i).Text)
        MinValue(i).Caption = RpcOleMin(i).Text
        MaxValue(i).Caption = RpcOleMax(i).Text
    Next
    RpcOleSTAQ.ElemName = Equipment
    RpcOleSTAQ.Property = "STAQ"
    RpcOleSTAQ.RefreshPeriod = 2000
    RpcOleSTAQ.Action = 3

    'AQN refresh rate (ms)
    TimerPeriod.Text = 500
End Sub
Private Sub HScrollVoltage_GotFocus(Index As Integer)
    ActiveProperty = Index
End Sub
Private Sub RpcOleCCV_GotFocus(Index As Integer)
    ActiveProperty = Index
End Sub
Private Sub HScrollVoltage_Change(Index As Integer)
    If OKBtnPressed = False Then
        RpcOleCCV(Index).Text = HScrollVoltage(Index).Value
        RpcOleCCV(Index).Action = 2
        RpcOleCCV(Index).Action = 3
    End If
    OKBtnPressed = False
    On Error Resume Next    'I.g. with different value ranges
    If OneVoltage And (Index = ActiveProperty) Then _
        HScrollVoltage(Abs(ActiveProperty - 1)).Value = _
            HScrollVoltage(ActiveProperty).Value
End Sub



Private Sub HScrollVoltage_Scroll(Index As Integer)
    On Error Resume Next    'I.g. with different value ranges
    RpcOleCCV(Index).Action = 0
    RpcOleCCV(Index).Text = HScrollVoltage(Index).Value
    If OneVoltage And (Index = ActiveProperty) Then
        HScrollVoltage(Abs(ActiveProperty - 1)).Value = _
            HScrollVoltage(ActiveProperty).Value
        RpcOleCCV(Abs(ActiveProperty - 1)).Text = _
            RpcOleCCV(ActiveProperty).Text
    End If
End Sub

Private Sub OKVoltage_Click(Index As Integer)
Dim NewValue As Integer
    If Val(RpcOleCCV(Index).Text) < 32767 Then
        NewValue = CInt(Val(RpcOleCCV(Index).Text))
    Else
        NewValue = HScrollVoltage(Index).Max
    End If
    If NewValue > HScrollVoltage(Index).Max Then
        NewValue = HScrollVoltage(Index).Max
        RpcOleCCV(Index).Text = NewValue
    ElseIf NewValue < HScrollVoltage(Index).Min Then
        NewValue = HScrollVoltage(Index).Min
        RpcOleCCV(Index).Text = NewValue
    End If
    RpcOleCCV(Index).Action = 2
    RpcOleCCV(Index).Action = 3
    OKBtnPressed = True
    HScrollVoltage(Index).Value = NewValue
End Sub



Private Sub RpcOleCCV_NewData(Index As Integer)
On Error Resume Next
    HScrollVoltage(Index).Value = CInt(Val(RpcOleCCV(Index).Text))

End Sub

Private Sub RpcOleSTAQ_NewData()
    LabelSTAQ.Caption = RpcOleSTAQ.Text
    If UCase(RpcOleSTAQ.Text) = "OFF-VAC. INTERLOCK" Then LabelSTAQ.ForeColor = &HFF&
    If UCase(RpcOleSTAQ.Text) = "ON" Then LabelSTAQ.ForeColor = &HFF00&
    If UCase(RpcOleSTAQ.Text) = "INVALID" Then LabelSTAQ.ForeColor = &HFF0000
End Sub

Private Sub MoreLess_Click()
    If MoreLess.Tag = "more" Then
        Height = Height + 800
        MoreLess.Tag = "less"
        MoreLess.Caption = " ^^"
    Else
        Height = Height - 800
        MoreLess.Tag = "more"
        MoreLess.Caption = "vv"
    End If
End Sub
Private Sub Timer1_Timer()
Dim ColorBuffer
    ColorBuffer = Shape1.FillColor
    Shape1.FillColor = Shape2.FillColor
    Shape2.FillColor = ColorBuffer

    'Read AQN value
    RpcOleAQN(0).Action = 1
    RpcOleAQN(1).Action = 1

    'Update AQNLabel
    AQNLabel(0).Caption = RpcOleAQN(0).Text
    AQNLabel(1).Caption = RpcOleAQN(1).Text
End Sub

Private Sub TimerPeriod_Change()
On Error Resume Next
    Timer1.Interval = Val(TimerPeriod.Text)
    TimerScroll.Value = Val(TimerPeriod.Text)
End Sub
```

```
Private Sub TimerScroll_Change()
    TimerPeriod.Text = TimerScroll.Value
End Sub

Private Sub IncrValue_Change()
Dim i As Byte
Dim NewValue As Long
On Error Resume Next
    NewValue = CLng(Val(IncrValue.Text))
    IncrValue.Text = NewValue
    If NewValue >= 1 And NewValue <= 100 Then
        For i = 0 To 1
            HScrollVoltage(i).SmallChange = NewValue
            HScrollVoltage(i).LargeChange = 10 * NewValue
        Next
    ElseIf NewValue < 1 Then
        IncrValue.Text = 1
    ElseIf NewValue > 100 Then
        IncrValue.Text = 100
    End If
End Sub

Private Sub IncrValue_Click()
Call IncrValue_Change
End Sub
```

## B.6 The control panel for the QS_REX Equipment Module

Special requirements made it necessary to redesign the control panel that was to be used for the QS_REX Equipment Module instead of reusing the one for the QS_NT Equipment Module. The AQN values were not to be read via a hot-link, but instead triggered by a timer. In addition, the two Equipment Modules have different names for their CCV and AQN properties.

From a CERN computer, the source code for this control panel can be found at
\\SRV1_ISOLDE\CTL\Contrl32\REXSteerQuad\Src\QS_REXctlpanel.vbp
(where \\SRV1_ISOLDE\CTL\ is usually mapped as O:\).

```
'Written by Trond Lieng
'March 2000
Option Explicit
Dim OKBtnPressed As Boolean  'Ensures only one CCV call when OK button is pressed

Private Sub Exit_Click()
    Unload QS_REXForm
End Sub

Private Sub Form_Load()
    Dim i As Integer, Equipment As String
    On Error Resume Next

    QS_REXForm.Caption = Command$
    QS_REXForm.Show
    OKBtnPressed = False

    Equipment = QS_REXForm.Caption

    'Initialisation of values
    RpcOleMin(1).Property = "MINV"
    RpcOleMax(1).Property = "MAXV"
    RpcOleCCV(1).Property = "CCV"
    RpcOleAQN(1).Property = "AQN"

    RpcOleMin(2).Property = "MINV_H"
    RpcOleMax(2).Property = "MAXV_H"
    RpcOleCCV(2).Property = "CCV_H"
    RpcOleAQN(2).Property = "AQN_H"

    RpcOleMin(3).Property = "MINV_V"
    RpcOleMax(3).Property = "MAXV_V"
    RpcOleCCV(3).Property = "CCV_V"
    RpcOleAQN(3).Property = "AQN_V"

    For i = 1 To 3
        RpcOleMin(i).ElemName = Equipment
        RpcOleMin(i).Action = 1

        RpcOleMax(i).ElemName = Equipment
        RpcOleMax(i).Action = 1

        RpcOleAQN(i).ElemName = Equipment
        RpcOleAQN(i).Action = 1

        RpcOleCCV(i).ElemName = Equipment
        RpcOleCCV(i).Action = 1
        RpcOleCCV(i).RefreshPeriod = 3000
        RpcOleCCV(i).Action = 3

        HScrollVoltage(i).Value = Val(RpcOleCCV(i).Text)

        HScrollVoltage(i).Min = Val(RpcOleMin(i).Text)
        HScrollVoltage(i).Max = Val(RpcOleMax(i).Text)

        MinValue(i).Caption = RpcOleMin(i).Text
        MaxValue(i).Caption = RpcOleMax(i).Text
    Next
```

```
    RpcOleSTAQ.ElemName = Equipment
    RpcOleSTAQ.Property = "STAQ"
    RpcOleSTAQ.RefreshPeriod = 2000
    RpcOleSTAQ.Action = 3

    Timer1.Interval = 500
    AQNPeriod.Text = Timer1.Interval
End Sub

Private Sub HScrollVoltage_Change(Index As Integer)
    If OKBtnPressed = False Then
        RpcOleCCV(Index).Text = HScrollVoltage(Index).Value
        RpcOleCCV(Index).Action = 2
        RpcOleCCV(Index).Action = 3
    End If
    OKBtnPressed = False
End Sub

Private Sub HScrollVoltage_Scroll(Index As Integer)
    RpcOleCCV(Index).Action = 0
    RpcOleCCV(Index).Text = HScrollVoltage(Index).Value
End Sub

Private Sub OKVoltage_Click(Index As Integer)
On Error Resume Next
Dim NewValue As Integer
    If Val(RpcOleCCV(Index).Text) < 32767 Then
        NewValue = CInt(Val(RpcOleCCV(Index).Text))
    Else
        NewValue = HScrollVoltage(Index).Max
    End If
    If NewValue > HScrollVoltage(Index).Max Then
        NewValue = HScrollVoltage(Index).Max
        RpcOleCCV(Index).Text = NewValue
    ElseIf NewValue < HScrollVoltage(Index).Min Then
        NewValue = HScrollVoltage(Index).Min
        RpcOleCCV(Index).Text = NewValue
    End If
    RpcOleCCV(Index).Action = 2
    RpcOleCCV(Index).Action = 3
    OKBtnPressed = True
    HScrollVoltage(Index).Value = NewValue
End Sub

Private Sub RpcOleCCV_NewData(Index As Integer)
On Error Resume Next
    HScrollVoltage(Index).Value = CInt(Val(RpcOleCCV(Index).Text))
End Sub

Private Sub RpcOleSTAQ_NewData()
On Error Resume Next
    LabelSTAQ.Caption = RpcOleSTAQ.Text
    If UCase(RpcOleSTAQ.Text) = "OFF-VAC. INTERLOCK" Then LabelSTAQ.ForeColor = &HFF&
    If UCase(RpcOleSTAQ.Text) = "ON" Then LabelSTAQ.ForeColor = &HFF00&
    If UCase(RpcOleSTAQ.Text) = "INVALID" Then LabelSTAQ.ForeColor = &HFF0000
End Sub

Private Sub Timer1_Timer()
Dim ColorBuffer, i As Byte
    ColorBuffer = Shape1.FillColor
    Shape1.FillColor = Shape2.FillColor
    Shape2.FillColor = ColorBuffer

    'Read AQN value and update AQNLabel
    For i = 1 To 3
        RpcOleAQN(i).Action = 1
        AQNLabel(i).Caption = RpcOleAQN(i).Text
    Next
End Sub

Private Sub MoreLess_Click()
On Error Resume Next
    If MoreLess.Tag = "more" Then
        Height = Height + 800
        MoreLess.Tag = "less"
        MoreLess.Caption = " ^^"
    Else
        Height = Height - 800
        MoreLess.Tag = "more"
        MoreLess.Caption = "vv"
    End If
End Sub

Private Sub AQNPeriod_Change()
    On Error GoTo TheEnd
    AQNPeriodScroll.Value = Val(AQNPeriod.Text)
TheEnd:
Err = 0
End Sub

Private Sub AQNPeriodscroll_Change()
On Error Resume Next
    AQNPeriod.Text = AQNPeriodScroll.Value
    Timer1.Interval = AQNPeriodScroll.Value
End Sub

Private Sub IncrValue_Change()
Dim i As Byte
Dim NewValue As Long
On Error Resume Next
    NewValue = CLng(Val(IncrValue.Text))
    IncrValue.Text = NewValue
    If NewValue >= 1 And NewValue <= 100 Then
        For i = 1 To 3
            HScrollVoltage(i).SmallChange = NewValue
            HScrollVoltage(i).LargeChange = 10 * NewValue
        Next
    ElseIf NewValue < 1 Then
        IncrValue.Text = 1
    ElseIf NewValue > 100 Then
        IncrValue.Text = 100
    End If
End Sub

Private Sub IncrValue_Click()
Call IncrValue_Change
End Sub
```

# References


[1] The CERN website: http://www.cern.ch (with permission).

[2] The ISOLDE website: http://isolde.cern.ch (with permission).

[3] "Radioactive beams drive physics forward" by Thomas Nilsson, CERN Courier Volume 39 number 10, December 1999.

[4] "Visual Basic® 5 for Windows® for Dummies®" by Wallace Wang, IDG Books Worldwide, 1997.

[5] "Hardcore Visual Basic" by Bruce McKinney, Microsoft Press, 1997.

[6] "The ISOLDE Control System" by A. Bret, I. Deloose, A. Pace and G. Shering. PS/OP Note 91-27, 19 August 1991.

[7] "Access to the Controls of the PS Machines from the Windows 95 Platform" by Ivan Deloose, PS/CO Note 96-53, CERN, 1996.

[8] "Recommendations for the use of fieldbuses at CERN" by Baribaud, G et al. (Working Group of Fieldbuses), CERN-SL-96-058 (CERN-ECP-96-011), 1996.

[9] "Sensor Networks and Communication" by Robert M. Crovella. (Section 88 from "The measurement, instrumentation, and sensors handbook", John G. Webster (editor-in-chief), CRC Press LLC and Springer-Verlag GmbH & Co KG, 1999.)

[10] "SINEC: Introduction to PROFIBUS on Industrial PC", Siemens AG, C79000-B8976-C068/5, 1996.

[11] "Instrumenteringsteknikk" by Odd Arild Olsen, Tapir, Trondheim, 1994. (In Norwegian.)

[12] "PROFIBUS Technical Description", PROFIBUS International (Order-No. 4.002), September 1999.

[13] "SIMATIC S5: ET200 Distributed I/O System (Manual)", Siemens AG, EWA 4NEB 780 6000-02c (Order No. 6ES5 998-3ES22), 1995.

[14] "applicom®- General documentation", applicom international, applicom on-line documentation: http://us.applicom-int.com/files/manual/APPLICOM.EXE, February 2000.

[15] "SIMATIC S5: ET200M Distributed I/O Device (Manual)", Siemens AG, EWA 4NEB 780 6006-02 03, 1999.

[16] "SIMATIC: S7-300 and M7-300 Programmable Controllers Module Specifications (Reference manual)", Siemens AG, EWA 4NEB 710 6067-02 01, 1998

[17] "Ladete Partiklers Fysikk: kompendium (1988/1994)" by R. S. Sigmond, Institute of Physics, Norwegian Institute of Technology, p. 3-11 (this chapter is in English), 1994.

[18] "Le Système de Contrôle de ISOLDE et REX" by Frank Locci, PS/CO/Note 99-28 (Tech.), CERN, 16 December 1999. (In French.)

[19] "SINEC: DP Programming Interface", Siemens AG, C79000-B8976-C071/02, 1996.

[20] "applicom® PROFIBUS", applicom international, applicom on-line documentation: http://us.applicom-int.com/files/manual/PROFIBUS.EXE, February 2000.

[21] ' - Operating mode' section of "applicom®- General documentation". (Ref. **14** above.)

[22] "Windows NT as Device Server for the ISOLDE-REX Project" by Ivan Deloose, PS/CO Note 97-27, CERN, 6 April 1998.

[23] "The ISOLDE Control System Reference" by A. Bret, I. Deloose, G. Leo, A. Pace and G. Shering, PS/OP Note 91-17, 6 October 1992.