

Politecnico di Milano

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Gestionale

CERN LIBRARIES, GENEVA



CM-P00071249

**THE CONTROL SYSTEM OF THE ISOLDE
FACILITY**

CERN LIBRARIES, GENEVA

Relatore: Chiar.mo Prof. Marco Somalvico

Relatore esterno: Ing. Alberto Pace

Tesi di Laurea di:

Giovanni LEO

Matr. Nr. 580195

Anno Accademico 1992-1993

Due & Johnson

25 OCT. 1993

POLITECNICO DI MILANO

**Facolta' di Ingegneria
Corso di Laurea in Ingegneria Gestionale**

**THE CONTROL SYSTEM OF THE ISOLDE
FACILITY**

Relatore: Chiar.^{mo} Prof. Marco Somalvico

Relatore esterno: Ing. Alberto Pace

Tesi di Laurea di:

Giovanni LEO

Matr. Nr. 580195

Anno Accademico 1992-1993

Summary.

The ISOLDE project consists of the move of CERN's Isotopes Separators and their experimental area from the recently de-commissioned Synchrocyclotron to a beam line served by the Booster Synchrotron. A new control system was required.

Traditionally, control systems for accelerators are based on software tailored to optimize potential utility and runs on sophisticated hardware. This results in "home grown" products which often remain incomplete and are overtaken by the rapid advances of the massive industrial base of commercial software.

The purpose of this thesis is to take the opportunity to explore the extreme opposite approach for control systems. Namely to use "market leader" commercial software and hardware available for the huge PC market, with in-house development limited to the Graphic User Interfaces and to the necessary hardware and software interconnects.

The results are positive, as the system proved to work correctly, and the physicist community is fully satisfied of its friendliness.

Table of contents

Table of contents	1
Acknowledgments.	5
1. Introduction.....	7
2. State of the art for accelerator control systems.	12
2.1. Introduction.....	12
2.2. The Reference Model.	12
2.2.1. The control room layer.	13
2.2.2. The front end computing layer.	14
2.2.3. The Equipment Control Assembly layer.	18
2.2.4. Networks.	18
3. The ISOLDE Facility.	21
3.1. Physics of ISOLDE.	21
3.2. Description of the beam transport system of ISOLDE.	22
3.2.1. Layout.....	22
3.2.2. The beam distribution system.....	23
3.2.3. The control system.	25
3.2.4. Numbers.....	25
3.3. The Research Problem.	26
4. Conceiving the ISOLDE control system: ISOLDE and the Reference Model.....	28
4.1. Architecture.....	28
4.2. The Control Room Layer.....	29
4.3. The Front End Computing Layer.	31
4.4. The Equipment Control Assembly Layer.....	33
4.5. Networks.....	33
4.6. Bus Standards.....	33
4.7. Strategies.	34

5. The implementation of the ISOLDE control system: the console level.	37
5.1. Experience in SPS and PS.....	37
5.2. The use of a "culture".	39
5.3. Which culture we implemented.	40
5.3.1. Analysis of the operations necessary to control ISOLDE.	40
5.3.2. Organization.....	41
5.3.3. Economization.	46
5.3.4. Communication.	49
5.3.5. Multiple Views.....	51
5.3.6. Color.....	53
5.4. How applications are implemented.....	57
5.4.1. Application structure.	57
5.4.2 The MainForm form.	59
5.4.3. The GLM form.....	64
6. The implementation of the ISOLDE control system: the Front End Level.	69
6.1. Introduction.....	69
6.2. Description of a QP.	69
6.3. The control of a QP.	73
6.4. Description of the properties.....	76
6.4.1. The MINV and MAXV properties.....	77
6.4.2. The SCLR, SCCR, SCLW and SCCW properties.	77
6.4.3. The CCV, ADRW and CHNW properties.....	78
6.4.4. AQN, ADRR and CHNR properties.	80
6.4.5. The STAQ, ADRD, CHNIO and CTRW properties.....	80
6.5. The QP equipment module.....	80
6.5.1. The function D_POWP.....	81
6.5.2. The special functions.	84
7. Obtained results and critical evaluation.....	86
7.1. How the system works.....	86

7.2. Example.	87
7.2.1. Initialization of the FEC.....	87
7.2.2. Initialization of the console.....	88
7.2.3. The information routing.....	89
7.3. Applications.	92
7.3.1. The Targets application.	92
7.3.2. The GPS, HRS, and Experimental Area.....	93
7.3.3. The QPArray application.	94
7.3.4. The Vacuum application.....	95
7.3.5. The GasMix application.....	96
7.3.6. The Faraday Cup application	97
7.4. Critical evaluation.....	98
8. Conclusions.	101
8.1. Simplicity.	101
8.2. Development time.....	101
8.3. Integration.....	102
8.4. Speed.	102
8.5. Cost.	103
8.6. Resonance.	103
References.	104

Appendix A : Some information about CERN.

Appendix B : Synthesis of the "Designing Graphical User Interfaces" tutorial.

Appendix C : The listing of the POWP.C equipment module and the HWACCESS.C functions.

Appendix D: Database files: EQPMOD.CSV, FECADDR.CSV, EQPNAMES.CSV, and POWP.CSV.

Appendix E : The ISOLDE Control System Reference.

Appendix F : Colored figures.

Acknowledgments.

I wish to express my thanks to:

Prof. Marco Somalvico, for the faith and independence he gave me on the orientation of this work, and for his precious advices;

Alberto Pace who, as my supervisor, have guided me through my thesis work, introducing to me the world of applied research, and always offering his generous disponibility;

Ove C. Jonsson, who assisted me in this project, for his efficient help, and his always good mood;

Ivan Deloose, who worked with me dividing the same office, and always contributed for a nice and enthusiastic atmosphere;

and my family.

1. Introduction.

The ISOLDE project (Ref. 21) consists of the move of CERN's Isotopes Separators and their experimental area from the recently de-commissioned Synchrocyclotron to a beam line served by the Booster Synchrotron. ISOLDE (Fig. 1.1.) is an on-line isotope separator facility where two mass separators supply radioactive ions, from the whole nuclear mass range, to about 250 physicists making experiments in the domain of atomic and nuclear physics, astrophysics, nuclear medicine, and surface and solid state physics (Ref. 22).

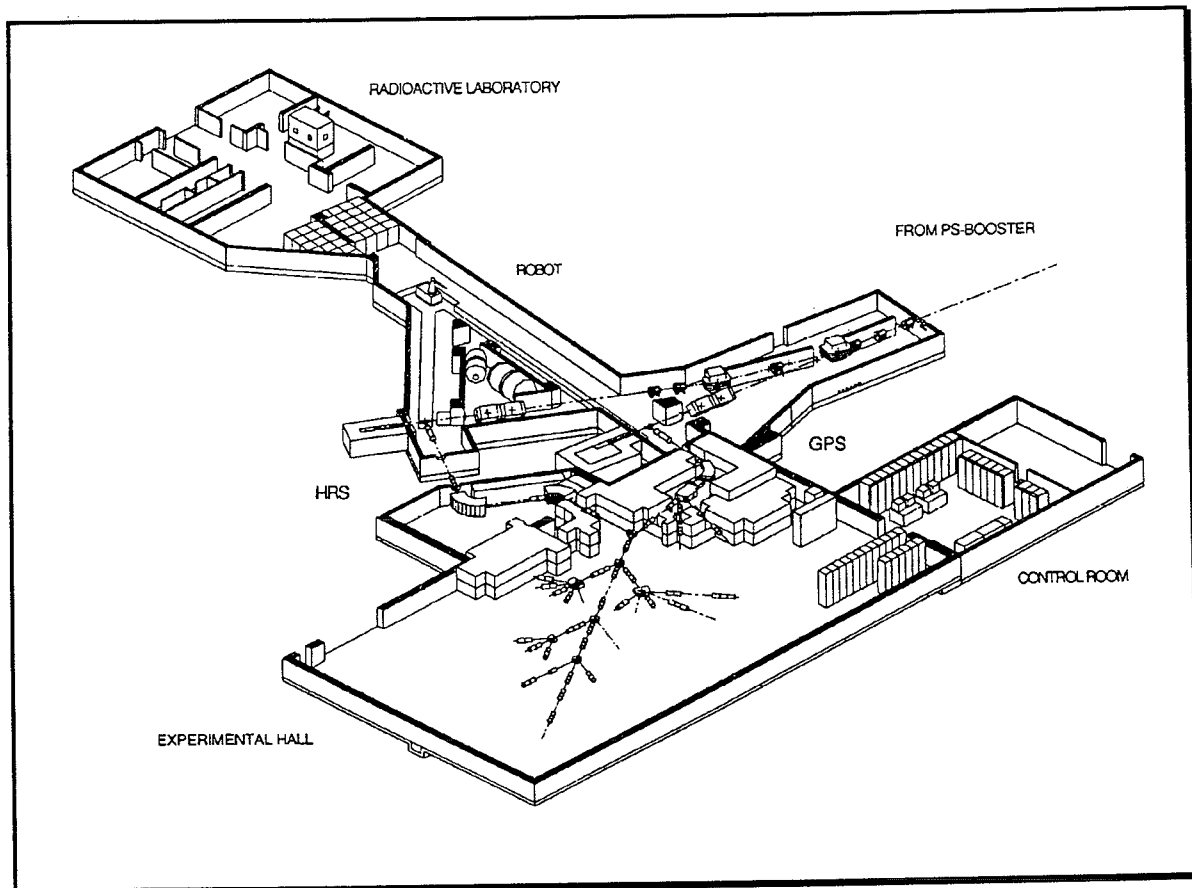


Figure 1.1. View of the ISOLDE Facility.

Part of the project consisted in developing a new control system (Ref. 23), with the only limitation of reusing the old CAMAC hardware. ISOLDE was commissioned by a

group of universities and research centers mainly interested in the physics they can do with the facility. Because of this, no extra budget to afford the costs of manpower to develop and maintain the control system was available. It was then clear that an inexpensive, user friendly, control system was needed.

Unfortunately, traditional control systems for accelerators are based on software tailored to optimize potential utility and run on sophisticated hardware. These types of control systems require many man-years of work of expert informaticians to be implemented, developed, and tested. Also their maintenance is very expensive requiring big amounts of time and money. Moreover, new applications and modifications are rare, due to the investment in time and money they require (Ref. 2,6, and 12).

It was then decided to implement a control system requiring a minimum of manpower both for the implementation and for the subsequent maintenance. The idea was that users of the facility, with little training, should have been able to maintain the control system, and to write and modify the applications.

The purpose of this thesis is to take the opportunity to explore the extreme opposite approach for control systems. Namely to use "market leader" commercial software and hardware available for the huge PC market, with in-house development limited to the Graphic User Interfaces and to the necessary hardware and software interconnects.

As a member of the group in charge of developing the system, my duties have been to propose a "culture" for the user interfaces, and once achieved, to implement it, and to build some application programs. In order to reduce the training period for the users to the minimum, many techniques to convey meaning to graphical representations have been tested and regularly evaluated in the course of formal and informal meetings we held with the users of the facility. I also worked in the development of some software needed to control the equipment of the facility, such as some elements that are called

quadrupoles, steering quadrupoles and deflecting plates. The function of quadrupoles is to focalize the ion beam, composed of charged particles, as it tends to drip. The deflector plates are elements that modify the trajectory of the beam. They consist of parallel charged plates connected to high voltage power supplies. The steering quadrupoles combine the functions of the two other elements, as they are able to focalize the beam and to slightly deviate it. To control this elements we insisted in the basic principle of using only massive commercial hardware and software. The control is then built by using commercial ISA bus cards, mounted on common PCs.

The results are positive, as the system proved to work correctly, and the physicist community is fully satisfied of its friendliness. In particular the time to develop the system has been strongly reduced, if compared with the time needed to develop similar systems. Moreover, goals such as cost, flexibility, upgrading possibilities, and maintenance, have been achieved.

The problem of implementing this control system has been attacked by making an inside analysis to see what were the needs (Ref. 13 and 15), and by making a survey of what the commercial products offered (Ref. 14, 16, 18 19 and 20). Once a trade-off between problems and solution was found, a test system was implanted and we began to write the software. Software is layered, so that functionality could have been tested before applications were finished.

Concerning the Graphical User Interface (GUI) aspect, as basis of the study, we considered the recent experiences done by two CERN working groups, that were responsible of the new control systems of the PS complex of accelerators and of the SPS ring collider respectively (Ref. 24 and 25). Some theoretical aids came also from a study on GUIs presented by Marcus Aaron, principle of Aaron Marcus and Associates, which is a consulting firm that designs and evaluates GUIs. A synthesis of this study is proposed in Appendix B.

In conclusion this is the structure of this thesis.

In section two the CERN Reference Model for control systems of accelerators is proposed. This technical report represents the state of the art for control systems. In parallel we discuss the characteristics of the available hardware and software alternatives for each layer of the system.

In section three we introduce the ISOLDE facility, we explain the physics that is done, and we describe the layout of the system.

In section four we describe the solution method. We show the conceptual architecture of the system and we explain how the ISOLDE control system was conceived layer, by layer.

In section five we describe into details how the upper layer (the console one) was implemented. After a discussion on how we conceived the graphic interface, in which we show also many pictures to explain how the concepts have been translated into reality, we give an example on how the applications were written, discussing the important parts of the code. As the discussion about the graphic interface consider colors, in appendix F, color copies of all the colored figures of this work are collected.

In section six we enter into details for what regard the Front End Computer (FEC) level. This section starts with the description of a type of element, as an example. Such element is a quadrupole, because it is simple in its working principles, and it profits of most of the relevant functions of the control system. The definition of the parameters and the hardware necessary to control the quadrupole follows, and the explanation of the software module written to control the element concludes this section.

Section seven describes the obtained results and explains how the system works. After an example on how an operator should work to control the previously discussed element, a critical evaluation of the project is reported, which describe good points and drawbacks of the solutions adopted.

Section eight reports the conclusions of this work. It points out what we believe are the most relevant characteristics that an easy and economical to built and operate control system should feature.

Section nine reports references while some appendices have been added. Appendix A gives some information about CERN, and appendix B contains a synthesis of the tutorial on GUIs that can be useful to better understand some concepts we introduced in the discussion of our graphic environment. Appendix C contains the C listings of the software modules necessary to drive a quadrupole element, and appendix D shows the contents of the database files. Appendix E contains the Reference Manual of the ISOLDE system and Appendix F color copies of the colored figures of this work.

2. State of the art for accelerator control systems.

2.1. Introduction.

In this section we will discuss a global architecture for modern accelerator control systems, based on International Standards, which was presented as a Reference Model for future control systems as a result of a survey conducted with the aim to make recommendations for rationalization in the major technological areas (Ref. 1), and we will review the characteristics and merits of the major components of that architecture.

2.2. The Reference Model.

Whatever the size of the project to be controlled, one may identify the man machine interface on one end and the equipment interface on the other. In between one finds a communication network, covering the distances, and some computer power to allow interaction between the operator and the equipment.

The study we said in the introduction has proposed a control architecture making, as much as possible, use of open systems conforming to International Standards.

This architecture is based on three computing layers, interconnected by two types of networks (Fig. 2.1.). The three layers are the control room layer with its consoles and central servers, the front end computing layer in the local areas, and the equipment control layer with its ECA's (Equipment Control Assembly) control crates which form part of the equipment.

Hardware and software used on each level are adapted to the specific functions of the layer, with a considerable diversity on the ECA level corresponding to the variety of accelerator components to be controlled. This approach differs from the one which was adopted for the old control systems where the same type of NORISK-DATA computers has been used for both (Ref. 2) control room and front end computing. The new architecture offers more flexibility and will permit continuous partial upgrading as

technology evolves, a feature which is essential in a time when computer power at a given cost doubles about every two years and when hardware and software standards keep changing fast.

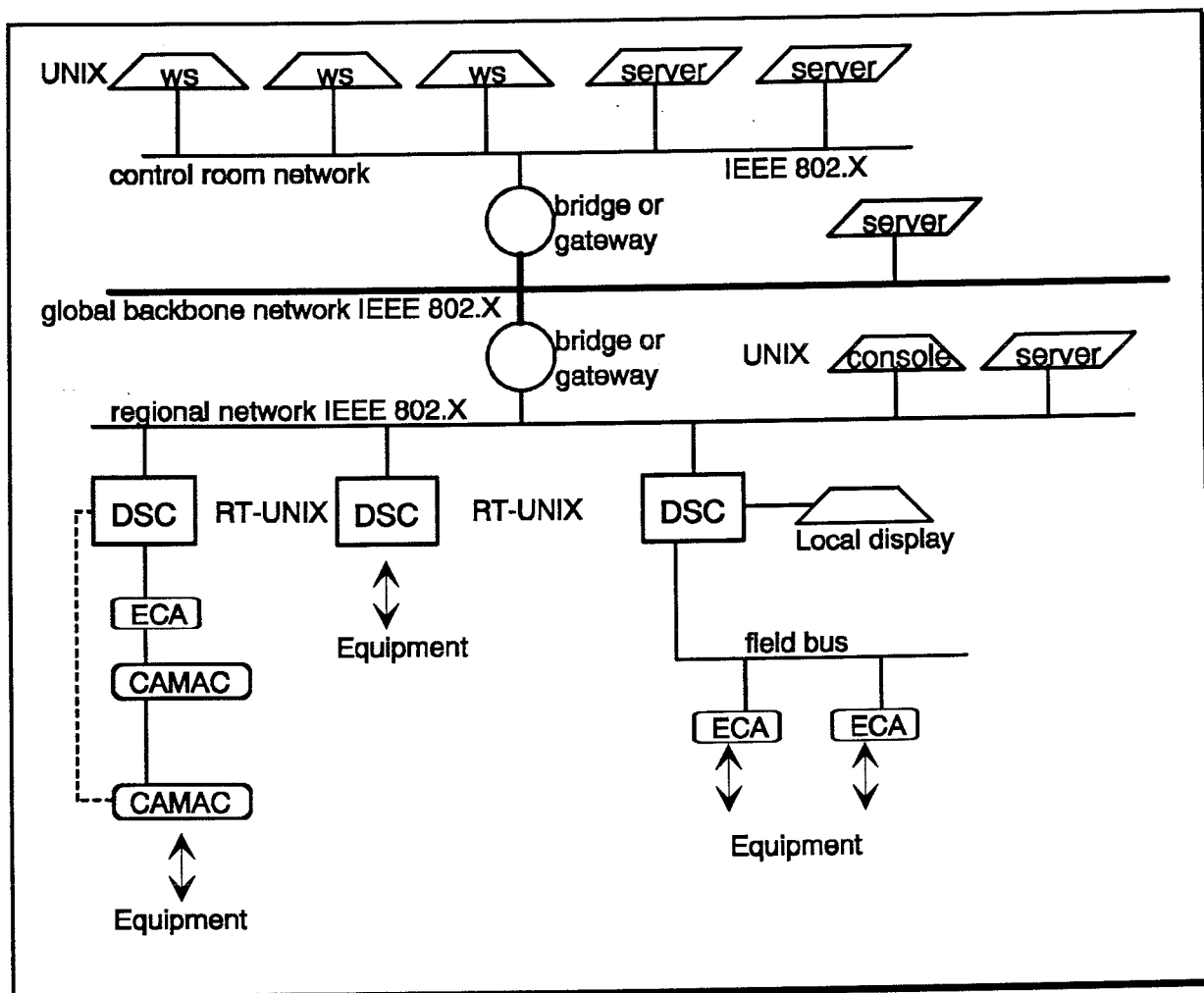


Figure 2.1. The Reference Model control system architecture.

2.2.1. The control room layer.

The control room layer has to fulfill two main functions. First of all, it must provide the operator with a reliable, user-friendly interface to the accelerators. As a second function, it must offer a number of central services, such as coordination of various control tasks, central data and file storage, model computing and collection of alarms.

To fulfill these requirements the UNIX environment has been adopted for the accelerator control systems at CERN (Ref. 3). In practice various versions of UNIX are used (Ref. 4): SCO XENIX 2.3 on Olivetti PCs, AIX 2.1 on IBM RTs, Domain OS 10.3 on Apollo, UX 8.05 on Hewlett-Packard and ULTRIX 4.2 on DEC workstations.

2.2.2. The front end computing layer.

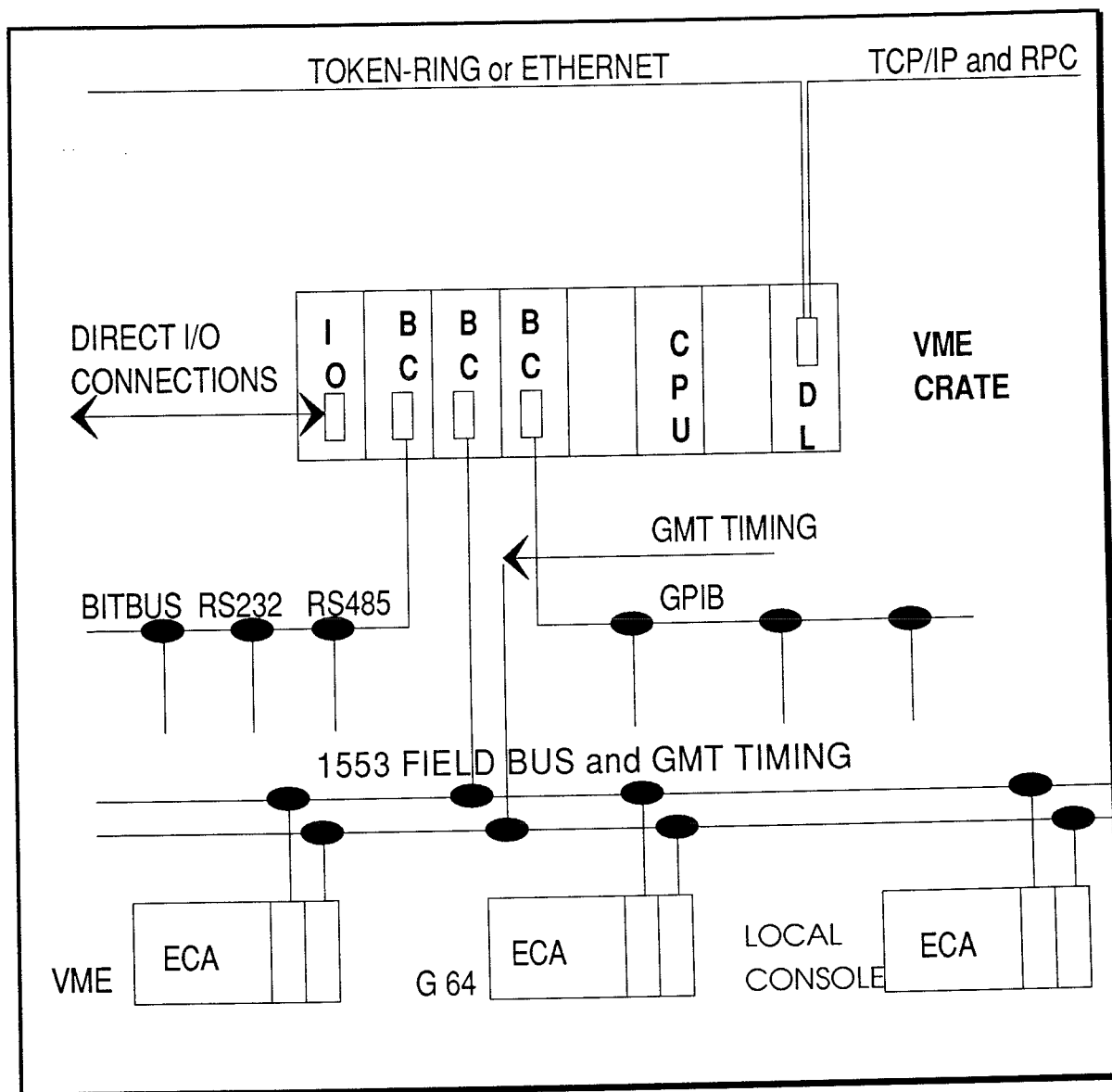


Figure 2.2. An example of a VME crate used as DSC at the front end level and connected at some ECA modules via the 1553 Field Bus.

The heart of the Front End Computing (FEC) layer are the Device Stub Controllers (DSCs) which are usually diskless, real-time UNIX compatible machines, but we will encounter some exceptions. Figure 2.2. shows a VME crate connected as DSC.

CAMAC was, in order of time, the first possible alternative as equipment to use as DSC (Ref. 5). It supports a high speed processor for control applications, and powerful modules for data acquisition. The first specification of the CAMAC system was finalized in 1969 in the EUR-4100 Document.

Following the success of CAMAC, more complex functions have been implemented, resulting in an emerging profusion of bus systems. Today the VMEbus is the most popular bus system (Ref. 6). The VMEbus with its functional modules is essentially an adaptation into EUROCARD format of the already existing Versabus from MOTOROLA. This is why VME is the acronym for "Versabus Modèle Européen". The VMEbus has also been accepted as IEC-821 standard.

The VMEbus has later been improved and expanded into a three bus combination VME-VSB-VMS (Ref. 7). The VSBus is a fast multiprocessor and memory subsystem bus, which uses 64 additional pins on the second dataway connector, while VMSbus provides a serial connection between modules for inter processor communication. The PC bus standards are the exceptions we mentioned above. The PC computer is usually a disk-based system and runs an operating system which is not UNIX. Despite this has been successfully used as a DSC in the LEP control system (Ref. 8).

The ATbus has been fully documented by two specifications, one from the IEEE Standard Organization (Ref. 9) and the other from Intel Corporation (Ref. 10). The generic name for the 8 bit and 8/16 bit AT compatible bus is Industrial Standard Architecture (ISAbus).

Recently, an Enhanced ISAbus (EISAbus) has been developed and specified by a manufacturer's consortium led by COMPAQ, (Ref. 11). The EISAbus provides a compatible 8/16/32 bit bus for the latest Intel 80386 and 80486 microprocessors. In developing the EISAbus specification the COMPAQ lead consortium added additional

features beyond the increase of the data width to 32 bits. Some of the features can be used by ISA platforms that only support 8 and 8/16 bit slots.

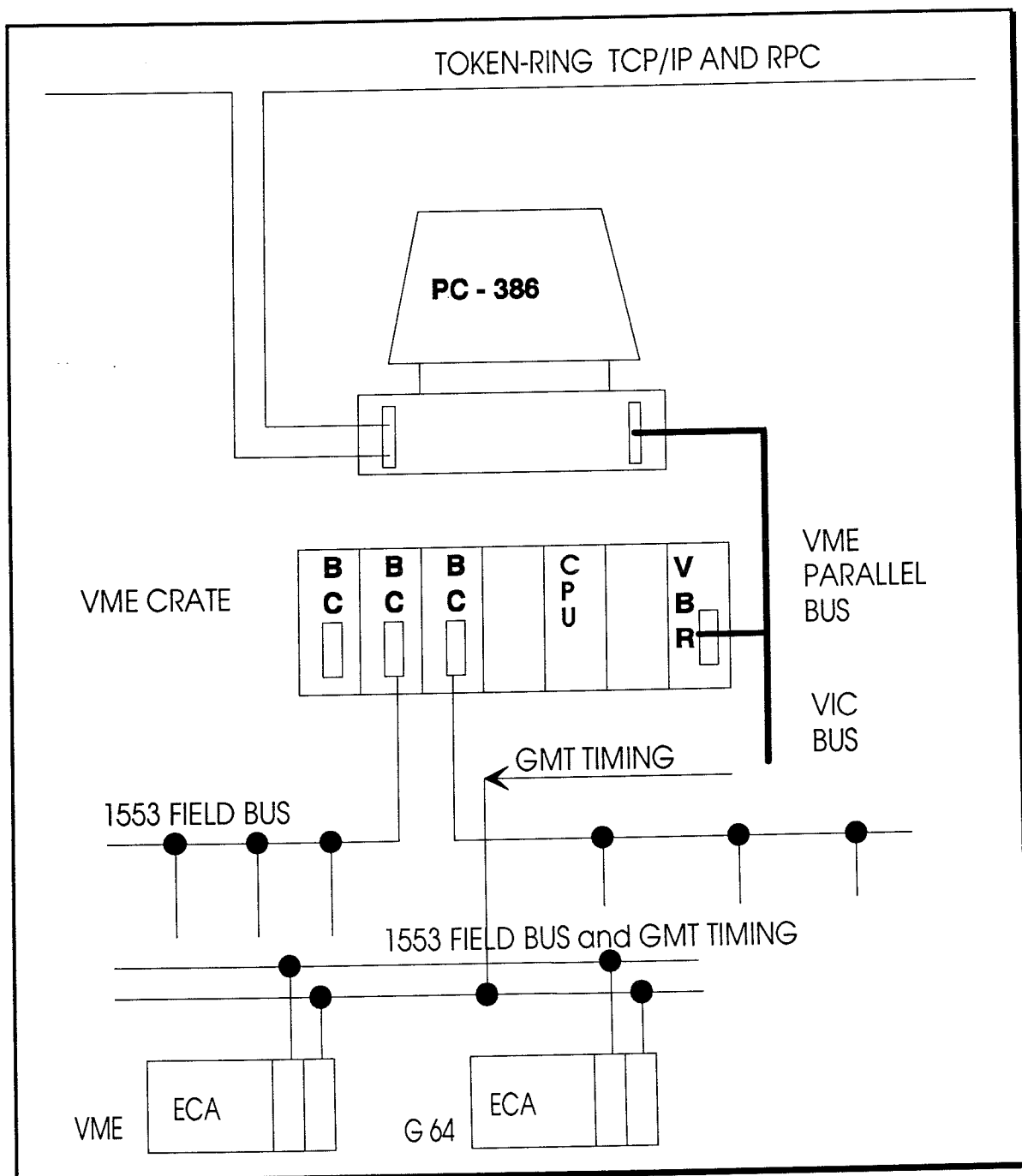


Figure 2.3. The VICbus used in a Process Control Assembly - PCA-.

To conclude our survey over the possible DSCs offered by the market we must introduce the VICbus standard (Ref. 12). In fact the widespread use of high performance, multi-processor systems based on backplane buses such as the CAMAC, the VMEbus, and also the PC buses has inevitably led to the requirement to create multi-crate systems. The VICbus (Vme Inter-Crate bus) is designed to achieve such assemblies in a standard way.

The development of a transparent multi-master multi-crate VME interconnection cable has started first at CERN in 1983. The objective was to use it for Process Control Assemblies (PCAs) in the control system of the future LEP accelerator, (Fig 2.3.).

The VICbus is a multiplexed, multi-master, multi-slave cable bus intended to connect multiple backplane buses or stand-alone devices. It provides transparent, softwareless interconnection for low latency short data transactions and fast data transmission of data blocks over cables of up to 100 meters in length. Address and data signals, each of 32 bits, together with those necessary for the control of the bus protocols, signal multiplexing, reset and error reporting are transmitted on twisted pairs using differential line drivers and receivers. Up to 32 devices are permitted on a single VICbus cable.

For what regard the operating system it is well known that UNIX is not designed to provide real-time features. For the front end process computers, a commercial real-time POSIX compliant operating system (LynxOS) has been chosen by CERN's accelerator control group (Ref. 13).

The acronym POSIX stands for Portable Operating System Interface and has been written by IEEE working groups and has been accepted as International Standard 9945, (Ref. 14).

MS-DOS operating system has to be remembered as it is the native operating system of the PC family.

2.2.3. The Equipment Control Assembly layer.

The Equipment Control Assembly (ECA) control crates are connected to the PCs or VMEbus crates via field buses on one side and to the equipment on the other side, by means of general purpose or dedicated electronic modules. Since no predominant standard exist, a certain variety of solutions, both for the ECAs and for the field buses, is to be expected and considered to be acceptable.

Their utility is to discharge work to the DSC. Some processes can be in this way computed in the ECA directly connected to the equipment and the DSC is only used to send commands and value requests. Possible ECAs are VME crates, local consoles and G64 modules (Ref. 15).

In the present three field buses are used at CERN: the 1553 field bus (Ref. 16), the GPIBbus (Ref. 17) and the serial CAMAC. (Ref. 18). In figure 2.2. is shown how some ECA modules are connected to a DSC via the 1553 Field Bus.

2.2.4. Networks.

As it is shown in Fig. 2.1. the Reference Model architecture is based on three computing layers interconnected by two types of networks. The communication within and between the Control Room and the Front End Computing Layers is based on modern Local Area Network (LANs) - Ethernet and Token-Ring conforming to IEEE 802.x International Standards and using the TCP/IP protocol. Where long distances (> 500 m) are involved a Token-Ring backbone network is implemented and transmitted (Ref. 19) over Time Division Multiplex (TDM) equipment conforming the CCITT G700 Standard.

To be as open as possible for future expansion, use of communication protocols conforming to the emerging ISO standards is recommended. Commercial products are available for the lower layers: the Physical layer, the Medium Access Control (MAC) layer and the Logical Link Control (LLC) layer as specified by IEEE 802.2. Figure 2.4. shows a schematic representation of the Network Protocol adopted.

For the upper layers, the de-facto standard is the US. Defense Advanced Research Project Agency (DARPA) suite of protocols. DARPA (Ref. 20) has defined the protocol for communication over an interconnected system of networks called TCP/IP (Transmission Control Protocol/Internet Protocol). This is exactly the framework of large control systems which are made from a number of Token-Rings and Ethernet segments interconnected by bridges, routers and gateways.

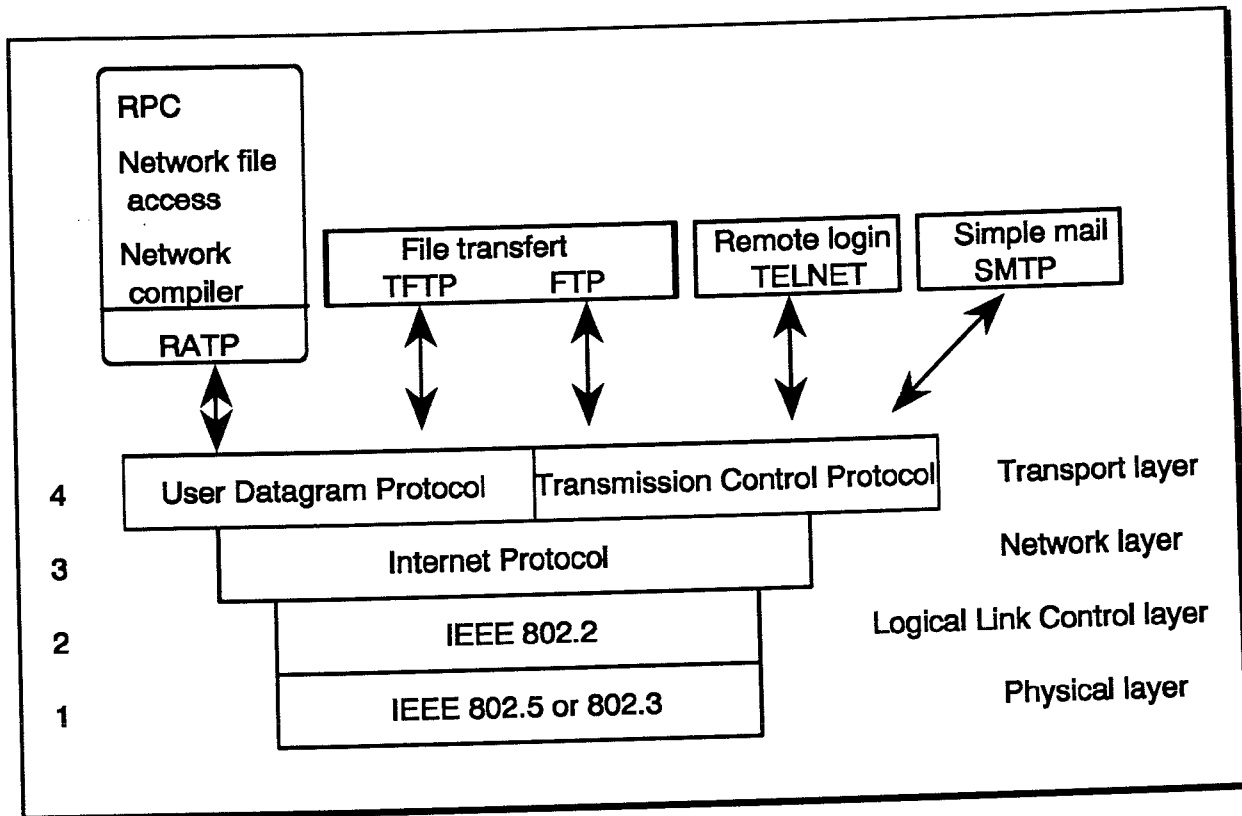


Figure 2.4. DARPA family of protocols (TCP/IP).

The DARPA suite of protocols is complete up to the Application layer and defines also a File Transfer Protocol (FTP), a virtual terminal utility via TELNET (TELEcommunication NETwork) and a Simple Mail Transfer Protocol (SMTP). The communication between the control center and the front-end process computers is achieved via the IP (Internet Protocol) and TCP or UDP. TCP (Transmission Control Protocol) offers to the user a fast and reliable connection-oriented protocol and UDP (User Datagram Protocol) offers a fast connection-less protocol with broadcast

capability, but with no guarantee that the message is delivered. All this network software is commercially available as portable packages written in C language.

On top of UDP, a Remote Procedure Call (RPC) mechanism offers a means for the programmer to call libraries located in remote computers as if they were available locally. The RPC software hides as much as possible the network to the user: it handles internal representation conversions, offers transparent remote calls and supports heterogeneous systems. A standard RPC is not yet available but IEEE and ISO standardization bodies are actively working on proposals and commercial products are expected to follow.

Such a network architecture offers a good separation of data traffic between various domain of activities. Typically, part of the data traffic on the control center network is separated from the backbone network and local traffic on the regional network does not interfere with the global communication.

3. The ISOLDE Facility.

3.1. Physics of ISOLDE.

Nature provides us with about 250 stable or very long-lived nuclei. The light nuclei, such as ${}^6\text{Li}$ and ${}^{12}\text{C}$, contain equal numbers of neutrons and protons whereas heavier nuclei have an excess of neutrons, for example, ${}^{208}\text{Pb}$ has 126 neutrons and 82 protons. A nucleus which contains its natural complement of protons or neutrons resembles a wet sponge. As more particles are added, it begins to drip. It is possible to create about 6000 artificial, radioactive isotopes, which spontaneously decay, drip, after a time which may be very short, characterized by the so-called half-life. Of these, some 2000 have been observed, many at ISOLDE.

Some of the very proton-rich isotopes, can be produced in several ways, for example by nuclear fission, but neutron-rich nuclei can only be created by violent, very high energy, interactions typical of ISOLDE (Fig. 3.1.).

In ISOLDE, a variety of radioactive reaction products are created in a thick target, by bombarding it with 1 GeV protons. After a sophisticated combination of high temperature chemistry, diffusion and ionization, the resulting isotopes are accelerated to 60 keV. A meticulous choice of target-ion source techniques, followed by mass separation in magnet are used to select and deliver isotope beams of the purity required by the experiments. Extensive research to develop target-ion units specific for each element has made about 600 different beams available, containing isotopes of over 60 elements from all regions of the periodic table, with intensities reaching 10^{12} ions/second.

ISOLDE supply today radioactive isotopes to about 300 physicists making experiments in the domain of atomic and nuclear physics, astrophysics, nuclear medicine, surface and solid state physics. As an example, an increasing number of ISOLDE users are benefiting from the almost unlimited choice of beams, to investigate

very low concentrations of impurities in materials. A major thrust is the study of impurities in semiconductors, which are of considerable technological and economic importance.

3.2. Description of the beam transport system of ISOLDE.

3.2.1. Layout.

The view of the new facility is given in figure 3.1. The protons from the PS-Booster (Ref. 21) are delivered to the ISOLDE targets zone via an underground transfer line, about 100 m long, coming from the left in the figure.

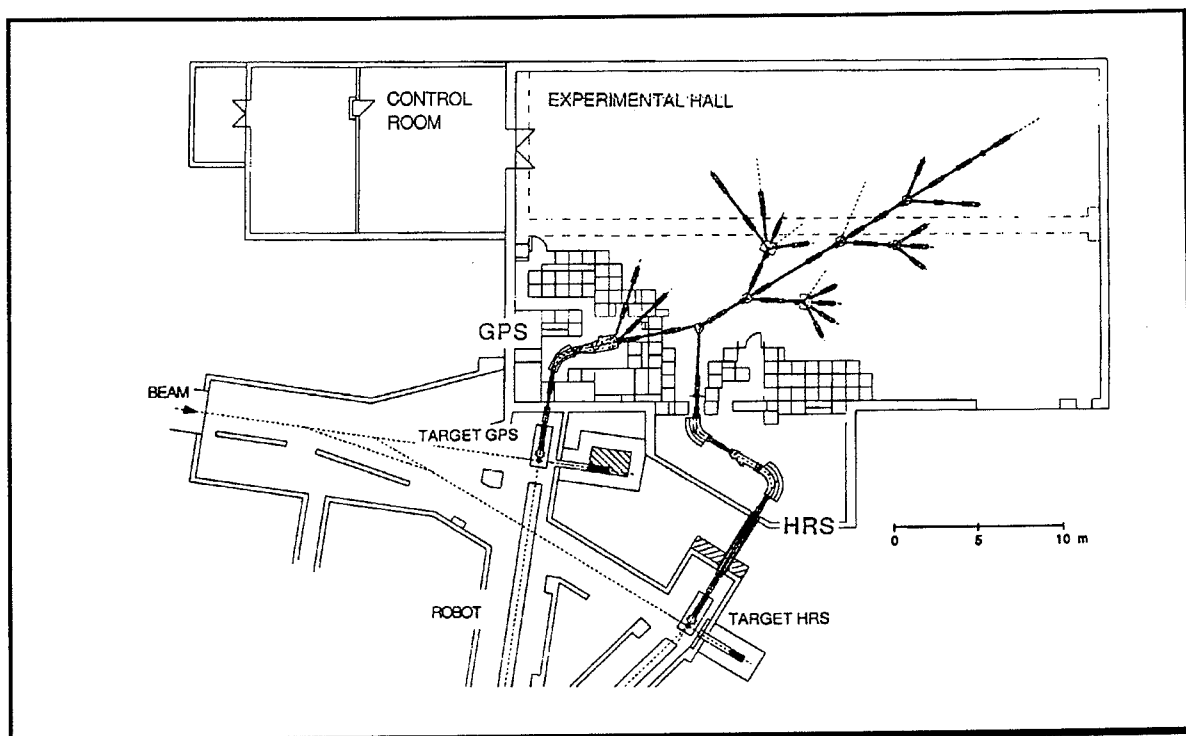


Figure 3.1. Layout of the ISOLDE Facility.

The beam can hit one of the two target stations which feed the General Purpose Separator (GPS) or the High Resolution Separator (HRS). The radioactive ions

produced in the target are accelerated to 60 keV, analyzed in the separators and then distributed to the experiments.

The GPS separator is easy to operate and versatile. It has one bending magnet and a switchyard of electrostatic deflector plates. HRS, with two bending magnets and an elaborate ion optical system gives a mass resolving power (a measurement of the purity of the separated mass component) 15 times better than GPS.

The two isotope separators are arranged such that a beam from either machine can be fed into a common beam distribution system to which almost all of the experiments in the 700 m² experimental hall are connected.

3.2.2. The beam distribution system.

A main objective in the design of the facility was to make best use of the available space, particularly of the experimental hall. To reach this goal a complex beam transport, elettrostatically driven, has been designed (Ref. 22). The most versatile mode of operation was to be able to supply the experiments with ions from either separator. To do this a "merging switchyard" has been constructed (Fig. 3.2.).

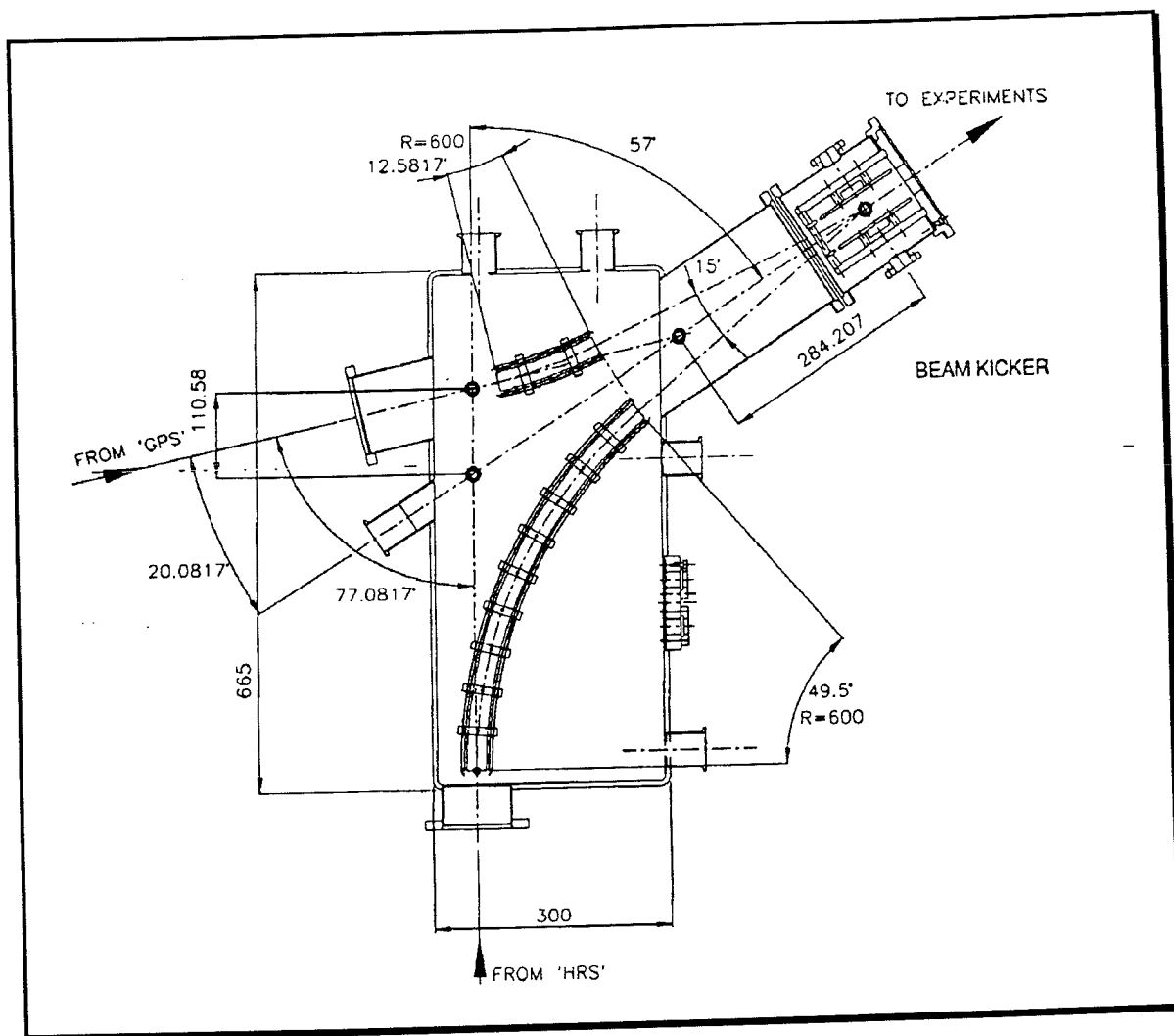


Figure 3.2. The merging switchyard.

Referring to fig. 3.2. we can see that the beam coming from the HRS or the central beam of the GPS are bent close to the final direction with the help of cylinder-shaped deflector plates. The final kick is given by a parallel plate condenser which can be excited with one or the other polarity. Since there are no moving parts in this "merging" switchyard, fast switching are possible, but obviously only one beam at a time can pass through it.

The beam distribution system is equipped with two five-way switchyards. Most of the experiments will be connected to these beam-lines. At the GPS two short beam-

lines will accept ions from the low and the high mass side of the mass spectrum. These two side beam-lines can be used in parallel with the central line.

3.2.3. The control system.

With the move of the ISOLDE facility to its new site a modern control system for both separators and the beam-lines had to be implemented. It was decided to have it based on inexpensive personal computers running under MS-DOS and Microsoft Windows and to make use of the CERN-wide installed network. The scope of this work is to implement this control system. Thus, next sections will develop in details the evolution of the system.

3.2.4. Numbers.

The implementation of a control system requires a first step in which analysis is done. The next step, the decisional one, can take great advantage from a correct analysis. In the case of ISOLDE analysis consisted in the determination of types and quantities of elements, in the determination of the number of analog and digital channels necessary to cope with them, in speed tests to guarantee a satisfactory throughput to the system, and in previsions about the manpower necessary to develop the system and about the cost of the system (Ref. 23).

The results of this step is relevant to understand the dimensions of the project and can be summarized as below.

The project can be quantitatively described as composed of:

- 2 Faraday cage 60 kV power supplies.
- 6 Extraction electrodes.
- 27 Faraday cups.
- 5 Slits.
- 5 Lens collimators.
- 15 Beam scanners.
- 5 Wire grids.

- 6 Thermocouples.
- 18 Target power supplies in 60 kV Faraday cage.
- 9 Beam gates.
- 3 Separator magnets with gauss meter.
- 58 Quadrupoles.
- 25 Steering Quadrupoles.
- 8 Kickers.
- 7 Benders.
- 4 Correction plates.
- 3 Multipoles.
- 5 Deflector.
- The Vacuum system.

This gives roughly 300 elements (devices) and 1700 analog or digital wires (control channels) entering the control system on three buses (CAMAC, GPIB, ISA).

More than 80 ISA (PC/AT) boards are or will be installed in different PCs or PC expansion crates. Eight FECs are then necessary, controlled by three consoles in the control room.

The manpower necessary to design the system was calculated in about 1.5 man*year, while another 1.5 man*year is necessary for the applications and the equipment modules.

Costs are equally divided between acquisition and computers. About 100 KCHF have been invested in PCs and the same amount in ISA boards, while 50 KCHF was the budgeted foreseen for cabling.

3.3. The Research Problem.

Trends in computer hardware and programming methodology are moving too fast for any one to present a comprehensive survey of ideas that might affect accelerator control systems. Nevertheless, there are several clearly identifiable developments which

started within the fairly recent past, continuing today, and which hold considerable promise for accelerator control in the near future. There are four major independent areas.

They are:

- The introduction of high performance, low cost workstations;
- Emerging network standards and network transparency.
- New software developments with respect to the human interface. Workstations incorporate many features that are a result of both software and human factors research. Among these are pointing devices and screen windowing systems.
- The emergence of efficient, integrated commercial database software, which can be used for all stages of an accelerator project, from early design, to construction, to the operational phase.

The challenge of this project is to design and build an accelerator control system that uses these powerful new developments in an appropriate and effective manner.

4. Conceiving the ISOLDE control system: ISOLDE and the Reference Model.

4.1. Architecture.

The ISOLDE Control system is simple. It has Olivetti personal computers (PC) at all levels connected using the general purpose Ethernet network available sitewide.

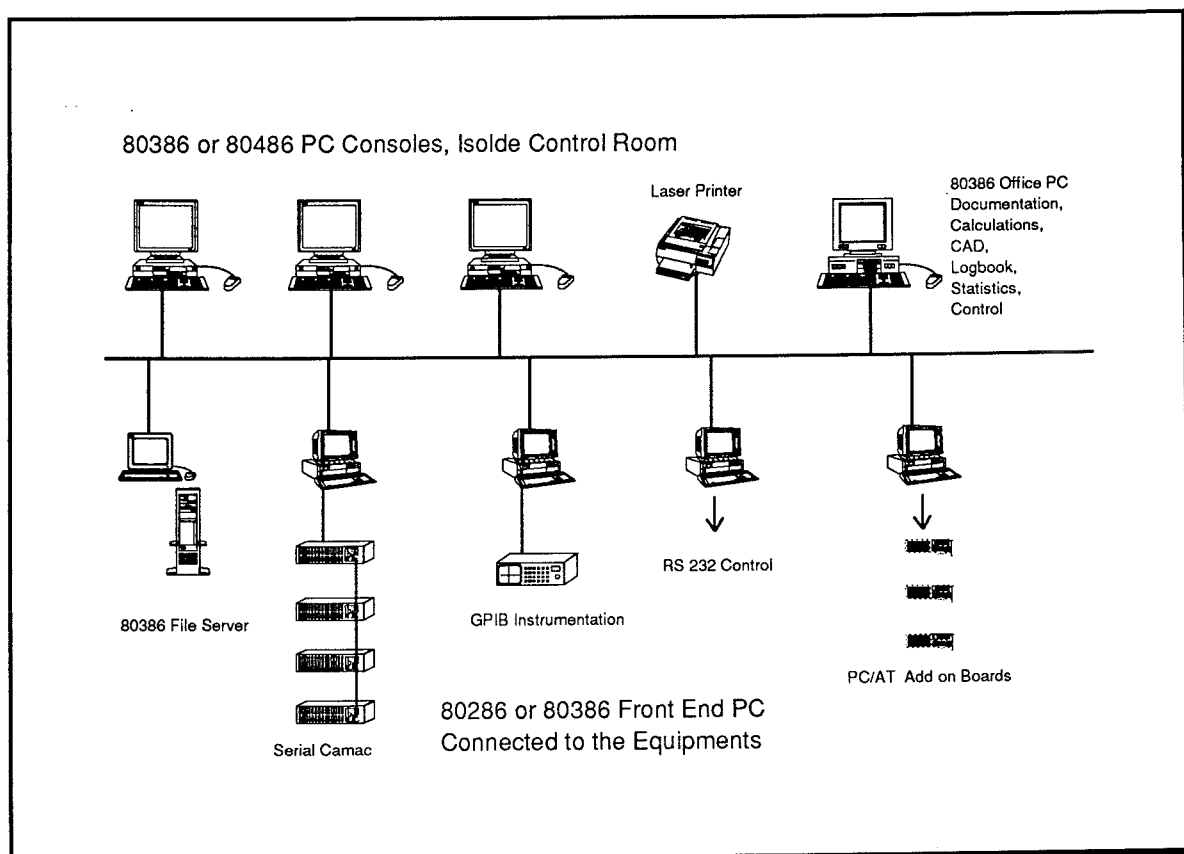


Figure 4.1. The ISOLDE Control System.

Figure 4.1. shows the architecture of the control system. The whole system is built on a commercial PC networking product from Novell called Netware. This network uses the protocol IPX/SPX (Internetwork Packet Exchange/Sequenced Packet Exchange), and provides a shared file system, shared printers, computer to computer

and computer to server communications that are used in the control system to share databases and the hardware equipment between different consoles. The Front End Computers are in fact just seen from the consoles as equipment servers to which client requests can be sent. It is indispensable to be attached to the ISOLDE server to execute any hardware access. This let the ISOLDE control system inherit all the security features that the Novell network provides.

4.2. The Control Room Layer.

The consoles in the control room are Olivetti 486/33 machines. The console computers are equipped with 21 in. color monitors giving a resolution of 1024*768 pixels.

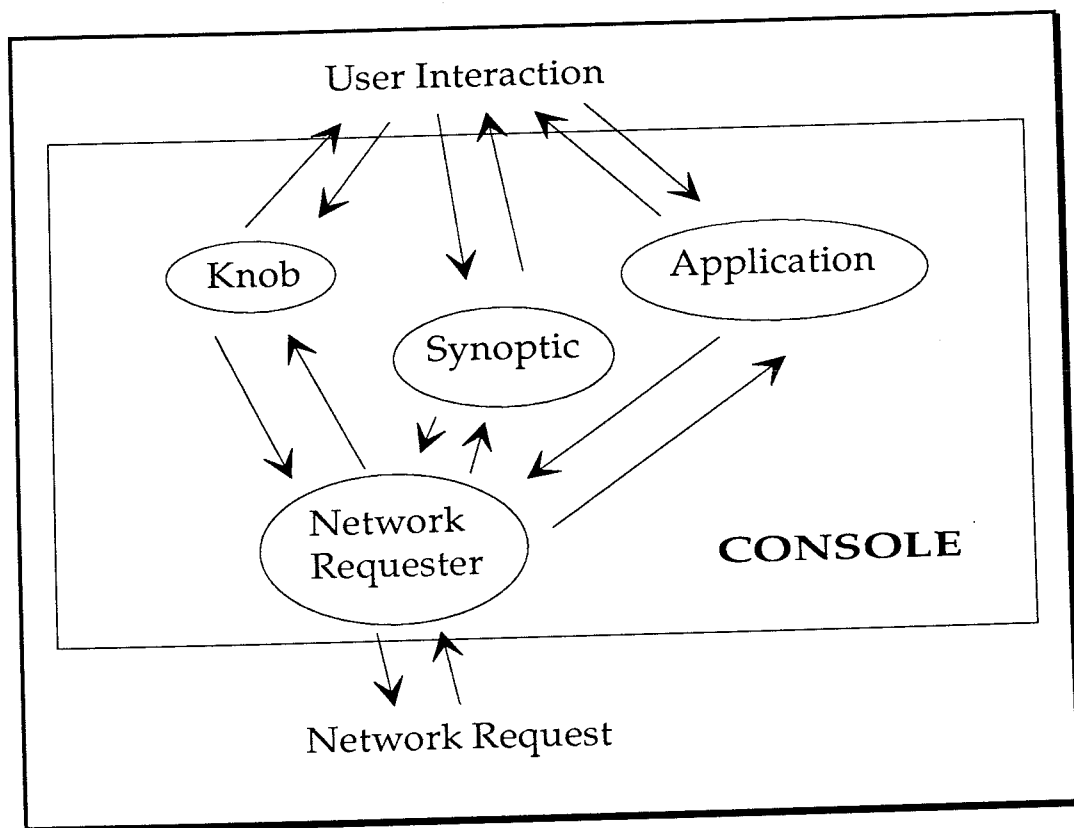


Figure 4.2. Data flow diagram showing how the software structure of a console works.

Besides the high resolution monitors, the consoles are fully compatible with all PCs available in the offices as local workstations. The Graphic User Interface (GUI) Microsoft Windows is a good complement to utilize properly the graphic performances of such PCs.

Figure 4.2 shows the data flow diagram of the software structure of the console. Application programs are written in C, using the Microsoft Software Development Kit™, Visual Basic™ and Excel™. Synoptics, the general purpose applications showing the whole area they are able to control, are all written in Visual Basic, due to its vocation for this kind of applications. Knobs are written in C to gain the maximum in speed, as we will expect to have many knobs on the screen at the same time, all competing for system resources. Applications, synoptics and knobs are shown in figure 4.3.

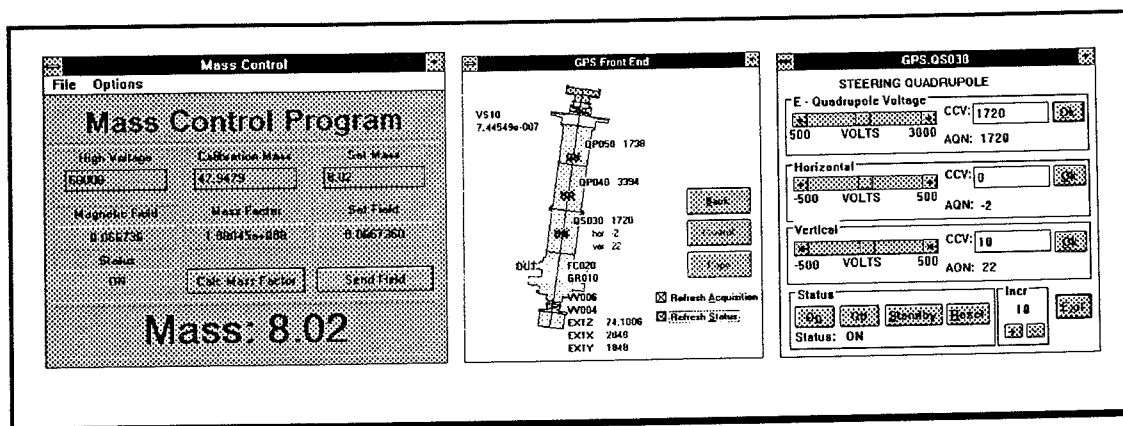
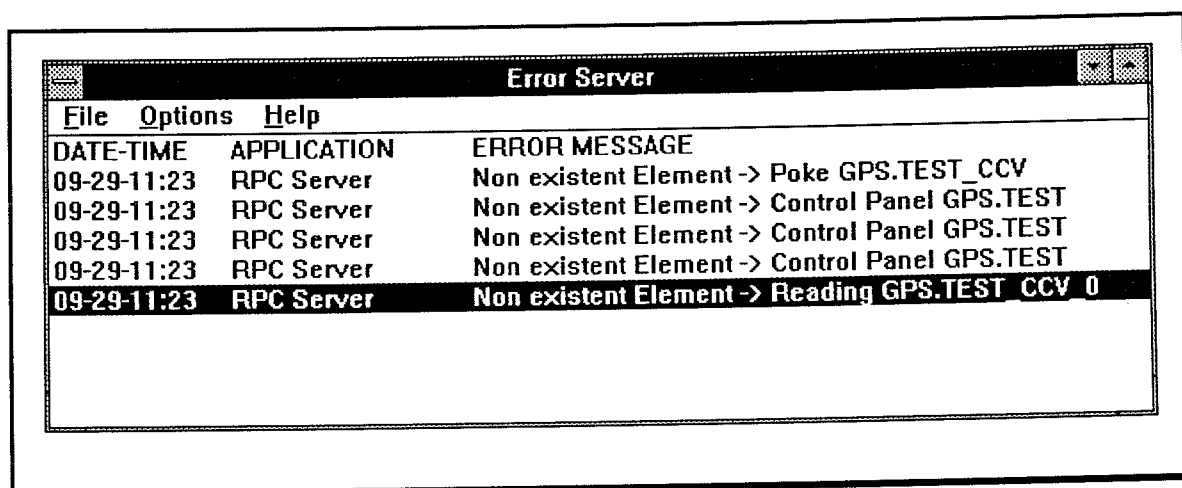


Figure 4.3 Example, from the left to the right, of an application, a synoptic and a knob.

The network requester, a C application named RPC-SERVER running in resident-mode on each console, collects all the requests from the applications running on the console and gives back to the client the requested information. Communications between applications and RPC-Server use the Dynamic Data Exchange (DDE) protocol. Every environment that supports Windows native Dynamic Data Exchange (DDE) protocol can be used to build-up applications. Among these we find the

Microsoft Windows Software Development Kit™ (SDK), Microsoft Excel™, Microsoft Visual Basic™, Microsoft Word™, Actor™, ToolBook™ and many others.

Communications between RPC-Server and FECs is done via a Remote Procedure Call (RPC) mechanism and the protocol here used is the Novell one, the IPX/SPX (Internetwork Packet eXchange/Sequenced Packet eXchange). An alarm process (Error Server) runs on all consoles and checks systematically for inconsistencies in the element names, properties, rights or that control values remains between defined ranges (Figure 4.4.). The alarm program in the console is normally minimized in a green icon that becomes red when active alarms arise.



The screenshot shows a window titled "Error Server" with a menu bar containing "File", "Options", and "Help". Below the menu bar is a table with three columns: "DATE-TIME", "APPLICATION", and "ERROR MESSAGE". The table contains five rows of error data, with the last row highlighted in black. The first row is a header, and the subsequent four rows show errors from "RPC Server" related to "GPS.TEST" elements.

DATE-TIME	APPLICATION	ERROR MESSAGE
09-29-11:23	RPC Server	Non existent Element -> Poke GPS.TEST_CCV
09-29-11:23	RPC Server	Non existent Element -> Control Panel GPS.TEST
09-29-11:23	RPC Server	Non existent Element -> Control Panel GPS.TEST
09-29-11:23	RPC Server	Non existent Element -> Control Panel GPS.TEST
09-29-11:23	RPC Server	Non existent Element -> Reading GPS.TEST_CCV 0

Figure 4.4. The Error Server.

4.3. The Front End Computing Layer.

The personal computers connected to the equipment are called Front End Computers (FEC) and are Olivetti 386/25 and old IBM 80286s. The performance of these computers is entirely satisfactory as they have enough CPU power to drive the equipment. The FEC PCs are identical in configuration to the Office PCs except that they have additional control boards.

hardware specific EM's give each FEC a different software configuration, which is loaded from the file server together with the data table at the startup of the FEC (Initialization).

4.4. The Equipment Control Assembly Layer.

The use of ECAs has been limited to the vacuum control system. They should close the vacuum valves if the pressure becomes too high, inhibit their control from the FEC if necessary, and start and stop pumping of given sections. Programmable Logical Controllers (PLCs) from Siemens have been chosen. This system offers a secure way to handle the vacuum system.

4.5. Networks.

As we saw the link between the layers consists of an Ethernet segment, connected via a gateway to the CERNwide Ethernet infrastructure. A flat backbone was suggested because of the simplicity of the system and the limited distances between the equipment and the control system. The file server of ISOLDE (an Olivetti 486/33 MHz) runs Novell Netware/386, a commercial PC networking product from Novell.

4.6. Bus Standards.

PC-CAMAC, PC-GPIB, PC-AT are the three types of busses that has been chosen to control the whole equipment:

- CAMAC was chosen because a wide range of standard modules is available and in order to allow the recuperation of the hardware from previous control systems.
- GPIB, a commonly used bus for measuring instruments, also known as IEEE-488 bus, was selected to drive sophisticated instrumentation.
- Industry Standard Architecture (ISA) boards that plugs into the PC directly or in ad hoc extension crates, were chosen for direct interfacing to the equipment. A

wide offer of this last type of boards exists and it has, for the moment, been restricted to Analog to Digital Converters (ADC), Digital to Analog (DAC), Digital Input Output (DIO), timers and RS-232.

4.7. Strategies.

The Man-Machine interface of the control system is based on a GUI. The Graphic User Interface is Microsoft Windows and provides a common access to all applications. The shell to start the different applications is the default Windows Program Manager (Fig. 4.6.).

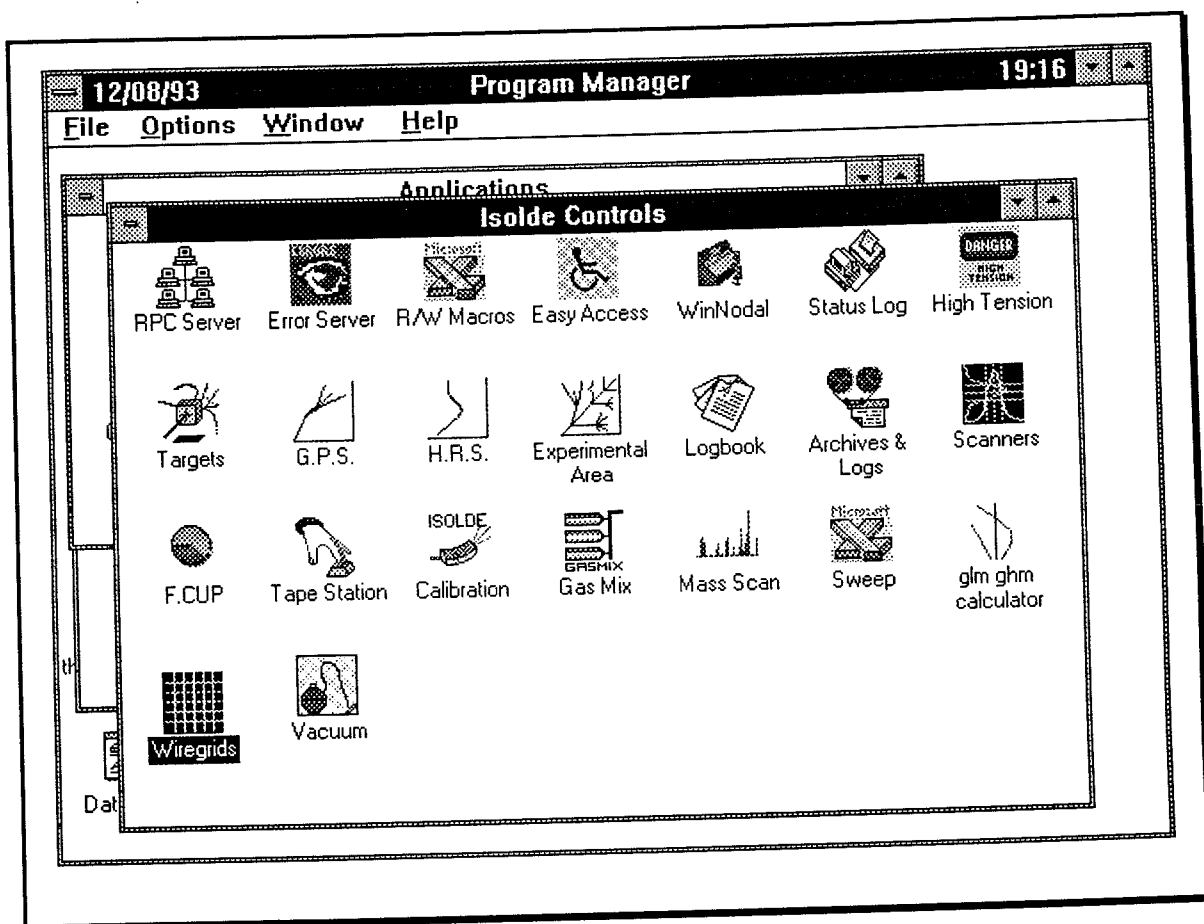


Figure 4.6. The Graphic User Interface.

The system is entirely database driven. Databases are built using Microsoft Excel, and stored in the native Excel format (BIFF) or in Comma Separated Value (CSV) formats when they must be accessible from applications (Fig. 4.7.).

The screenshot shows the Microsoft Excel interface with two spreadsheets open. The first spreadsheet, MPOWP.CSV, contains a table with columns A through H. The second spreadsheet, MPOWPDOC.XLS, contains a table with columns A through H, starting from row 2.

MPOWP.CSV								
	A	B	C	D	E	F	G	H
1	%	Database for MPOWP Equipment Module						
2	EqpNumb	Name	FecName	MINV	MAXV	MINV2	MAXV2	MINV3
3	1	GPS.QS5	POWER1	500	3000	-500	500	-500
4	2	GLM.QS7	POWER1	500	3000	-500	500	-500
5	3	GHM.QS7	POWER1	500	3000	-500	500	-500

MPOWPDOC.XLS								
	A	B	C	D	E	F	G	H
2	MPOWP Multiple Value Power Supplies access for the							
3								
4	Analog (Standard)							
5			INIT	R/W	ADCCurrent the same on all poles			
6			CCV	R/W	ADC Current in Amps			
7			CCVD	R/W	ADC in Bits			
8			AQN	R	DAC Current in Amps			
9			AQND	R	DAC in Bits			
10			CCV2	R/W	ADC Current in Amps			

Figure 4.7. One Excel Database.

This provides a means to store, in an organized structure, all static and dynamic variables. All parameters, scaling factors, node addresses on the network, default parameters, card and channel addresses, crate numbers etc., are stored in the database. This keeps the system maintenance to a strict minimum.

Excel also provides spreadsheets to calculate and display physical parameters of the separator, graphic facilities to summarize several parameters in one single chart and a

powerful macro language to write end-user applications used by the operation personnel.

5. The implementation of the ISOLDE control system: the console level.

5.1. Experience in SPS and PS.

ISOLDE is not the first control system that uses consoles with the potential to display a large number of windows. Some experience has been achieved at CERN from the groups responsible of the rejuvenation of the SPS collider [Ref. 24] and the PS complex [Ref. 25]. This experience consists mainly in the several iterations that were necessary to satisfy the operators. In fact, the powerful environment of these control systems permit the operators to run several tasks on a single machine, when they previously had to monopolize two or even three consoles. The drawback is that it also allows the operators to cover the screen with a multitude of windows, leading to complete confusion.

Both groups passed from a context made of fixed screens and fixed commands, to a context remarkable for the flexibility and the power of the consoles. For what regards SPS [Ref. 24], the first requirement adopted to develop the software considered the functionality of the system as a whole. Little time was first paid to the user interface. The programs were layered and this permitted them to develop user interfaces straight forward. Without going into details, and taking into account only the problems that are inherent also with the ISOLDE project, we can summarize as below the points experienced by SPS:

- With even a small number of windows, the screen can become very quickly confused.
- Some tasks involve a number of windows and it is often difficult to associate them with each other.
- Although modern window/graphics environments can deal with changes to window sizes, there is often an optimum, or at least a minimum useful window size for a

given application. If, as often happened, the operator accidentally changes the window size, time would be wasted re-setting it.

- On-line help information is of limited usefulness. They noted that the operators would often not bother to read detailed instructions on how to use the software, but would instead call the system developers to get an answer.
- As long as the error reporting system presented much more information than had often been the case with existing SPS software, the operators would be able to resolve many problems themselves. However, in practice, they often immediately called the system developers without reading the error message. This happened even when the reported error referred explicitly to equipment hardware problems.
- Many of the concepts in the system were not easily understood by the operators despite training. A typical problem was, for example, the trim software. When trimming the current function in a magnet system, one could choose from seeing the function as it presently stood, (the reference) or the difference from the theoretical current function (the trim). In practice, the operators were often confused by the distinction.

In addition to these observations, there were a number of issues that needed to be resolved. For example, the software design allowed the possibility of running multiple copies of various applications. As a desirable feature this would obviously require some management. Moreover the operators found the extreme flexibility of the screen layout to be slightly disturbing, and there was a requirement to produce a consistent "look and feel" of applications, covering such aspects as standard colors, positions and labels of icons performing similar actions, behavior of "pop-ups," menu styles and mouse button use. With this in mind the SPS re-wrote the interfaces and developed a console manager.

PS and his rejuvenation project [Ref. 25], started his work later and took benefits from the SPS experience. In this case a lot of work was done in the direction of giving a structure to the system. They proposed a system with a structure Beam Oriented rather than Hardware Oriented, and they introduced the concept of Application as the

environment needed to operate a logical part of an accelerator in an autonomous manner. They classified then tools contained in the Applications according to the different stages of a process. They have selected:

- Synoptics, used to visualize the whole process covered in the Application.
- Individual controls, needed to access each piece of hardware in the Application.
- Physical Parameters, introduced to hide some variables like Voltage, Intensity, and so on, and to give directly the relevant value to the physicist, like meters, radians, gauss...
- Measurements, having no special structure, due to the particularities of each process.

Conclusions presented from this group reported that the system was readily accepted by the operation team without a long training period. This positive reaction made us consider to extend those principles in the implementation of the ISOLDE Graphic User Interface (GUI).

5.2. The use of a "culture".

The advice we can keep from this experiences is that controls tend to be more and more sophisticated while operators need to work with simpler user interfaces. This could seem hard to achieve, but we need just to give a look on the huge market of commercial software to find a solution to this problem. The solution consists in introducing a "culture" on a family of software. This culture, if consistent, will be assimilated by the behavior of the user, who will keep it as a part of his knowledge.

To obtain this we must keep in mind that the success of this kind of solution is directly linked with two aspects of the culture which are: the culture needs to be consistent, and it needs to be uniformly spread out through the system.

In the PC world, Microsoft as an example, gives us several examples of culture spread on applications. In the Windows environment, if you buy a new application, you have in advance an idea on how commands will be interpreted. You know, for

example, that [CTRL]+[INSERT] will let you load in a buffer named "clipboard" the area of text, bitmap or vectorial image you have selected, and [SHIFT]+[INSERT] will let you insert the clipboard content, in the current application, starting from the position of the cursor.

In our case, what was to be done was to obtain a consistent user interface by taking into consideration both theoretical and empirical aspects. The theoretical approach took into consideration a recent study of Aaron Marcus, who is the principle of Aaron Marcus and Associates, a Californian consulting firm that designs and evaluates GUIs, and which is condensed in Appendix B. The empirical approach considers carefully the experience done by the groups responsible of the rejuvenation of the control systems of SPS and PS, and the many conversations had with the future users of the ISOLDE facility. In appendix F all the figures proposed are reported with colors.

5.3. Which culture we implemented.

5.3.1. Analysis of the operations necessary to control ISOLDE.

The implementation of every control system, requires some preliminary analysis to understand in details which and in which order functions need to be exploited, and how operators would like to conduce their job. For this reason few meetings were held before starting the implementation of the ISOLDE GUI, and some others were held to show and discuss the evolution of the project. In our particular case the future operators on the consoles will be the users of the facility, so they will be physicists and engineers.

The result of these meetings was a better comprehension of the facility. Its scope is to generate, separate and transport ion beams. The quality of the beam is determined by its composition and its spatial distribution. The control system does not affect the composition of the beam (the separator does it), but it is responsible for its spatial distribution.

The operator would like to control the direction and the spread of the beam up to the experiment. The direction and the focusing of the beam can be modified by means of electrostatic deflectors and lenses, due to the positive charge of the beam. The control of these deflectors and lenses is done by changing the potential of the electrodes that constitute the elements. The operator needs to have full control of the elements that are on the way of the beam.

In order to "see" the beam some detectors are placed on the way of the beam. These can be destructive or not. A detector is destructive if, when in place, stops the beam. The operator need then to be able to place and displace the destructive detectors, activate or not the others, and read the information given by the detectors in the most convenient representation.

This is the minimum a control system should provide. But, for a modern control system, some other functions are considered as a must. A modern control system should provide the status of the machines, a complete error report system, database functions for archiving, possibilities to store and retrieve configurations, report generation facilities, modeling, etc.

5.3.2. Organization.

To describe the implementation of the GUI is practical to follow the set of principles given in appendix B. The first principle is organization. The concepts related to organization are consistency, screen layout and navigability.

For what regards consistency we considered that Windows is a highly structured interface, and any Windows user is familiar with the behavior of the system. To confer external consistency to our structure we decided that programs should have followed the Windows standards as far as possible. The basic idea was that the most a program follows the Windows standards, the best was for the operator.

In this spirit we decided to create two new groups, the first containing the programs, the second the databases. Then we introduced the concept of application. An application should consist by one or more programs, be represented by one icon in

the applications group, and should contain the code necessary to operate a physical part of the system in an autonomous manner. The icons always shows a strong relationship with the physical part controlled by the associated application. The introduction of the concept of application fulfills the needs of navigability in the system and the strict relationship between hardware and software helps to fulfill requirements of real-world consistency. Figure 5.1. shows the content of the groups we created: the ISOLDE Control and the Databases groups.

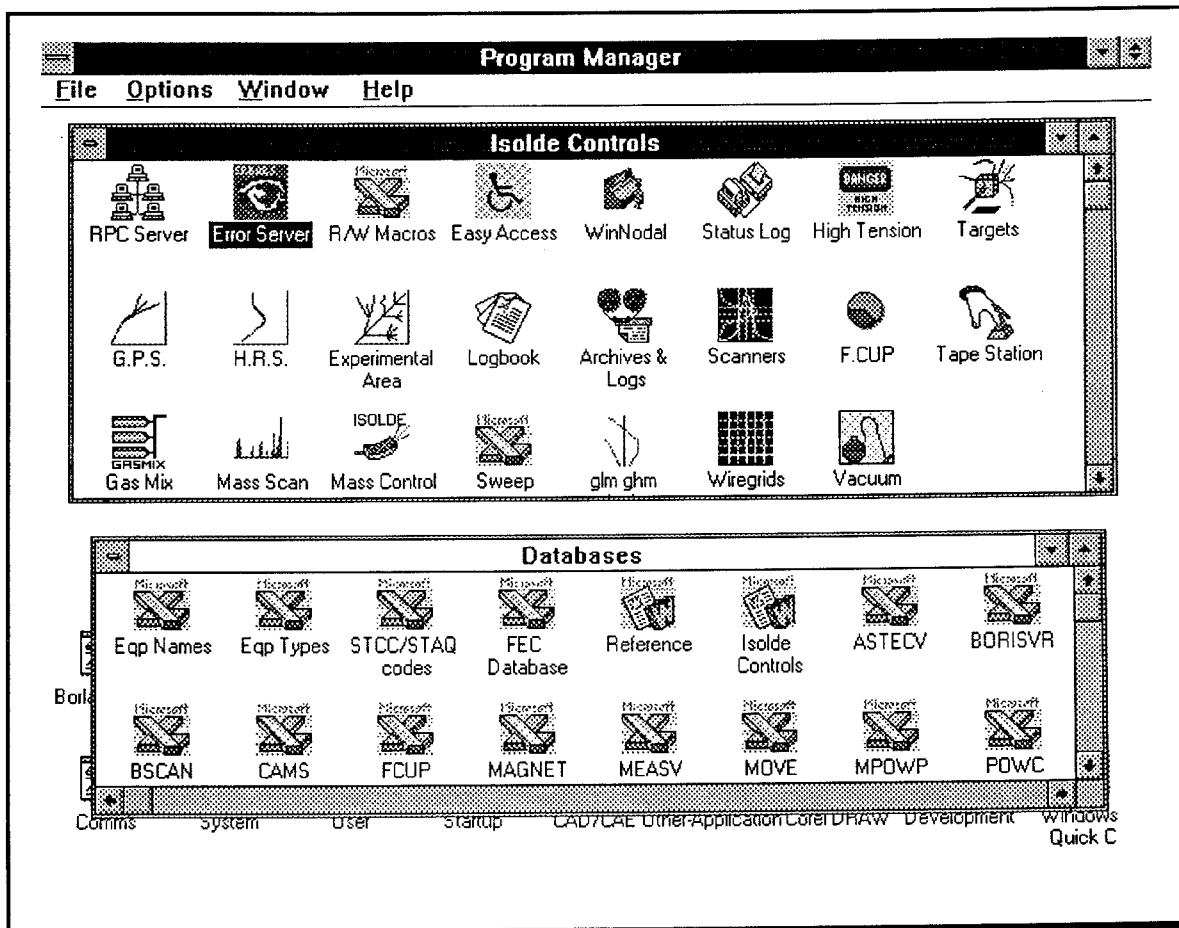


Figure 5.1. The ISOLDE Controls and Databases groups.

Screen layout organization concepts suggested that applications, once called, should open always in a fixed and known position, also if the user is then able to move them. The size of each window is also fixed and depends on the amount of data it should

show. In particular the size of each window is kept to the strict minimum, this to minimize, as far as possible, problems of screen competition.

To add real-world consistency the control of the system is kept Beam Oriented. This permits also to simplify the logic of the system and, as a direct consequence, the training. The beam network has been divided into three Synoptics. Each one gives access to a physical part of the beam network and the representative icons of these three applications specify clearly to which part of the network they are referring at (Fig. 5.2.).

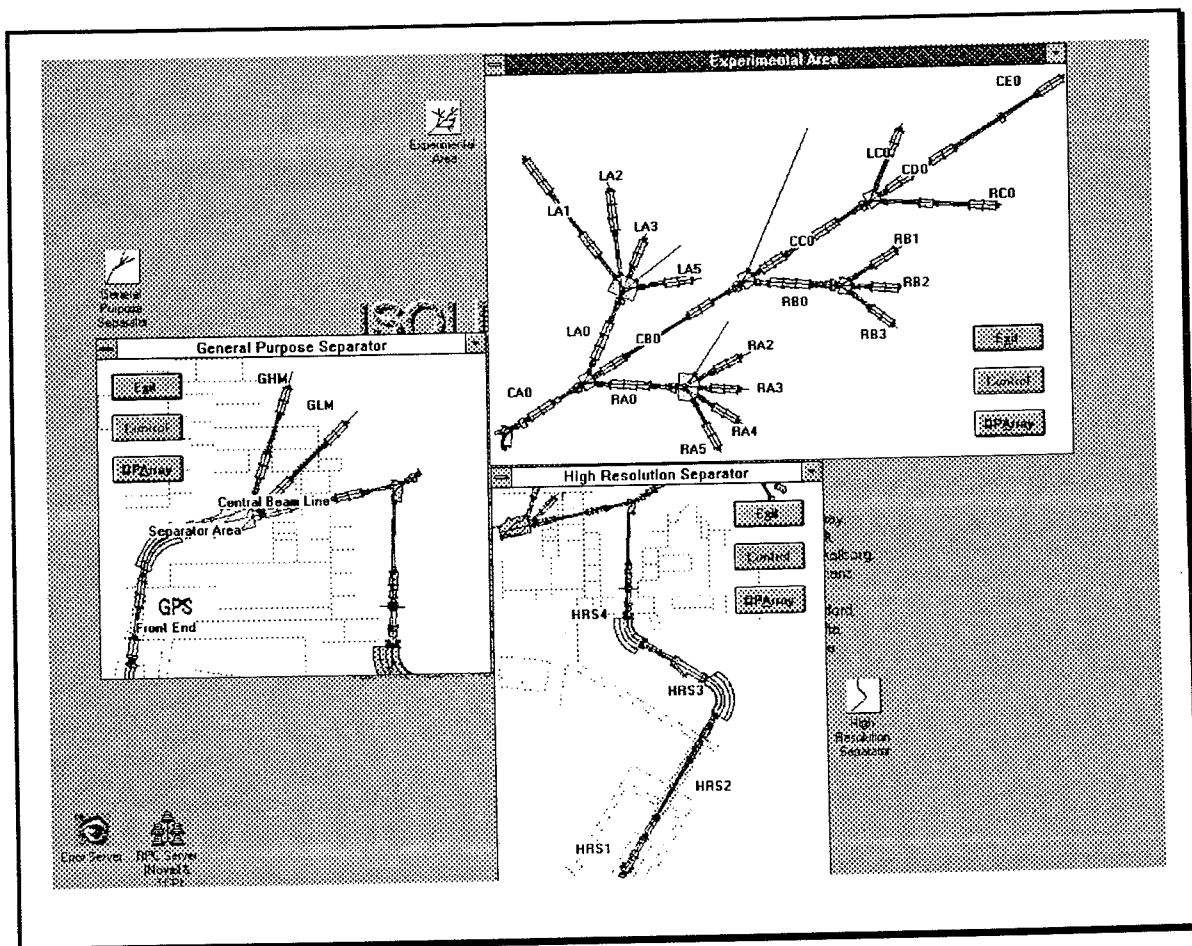


Figure 5.2. Complete view of the beam network and related icons.

This division is introduced with the aim to reduce the number of zooms necessary to control the system, the number of windows on the screen and to consequently improve the speed of the system, as the loading of a picture requires some seconds. The

division structure follows the schedule of the project, as the three sections are implemented at different stages.

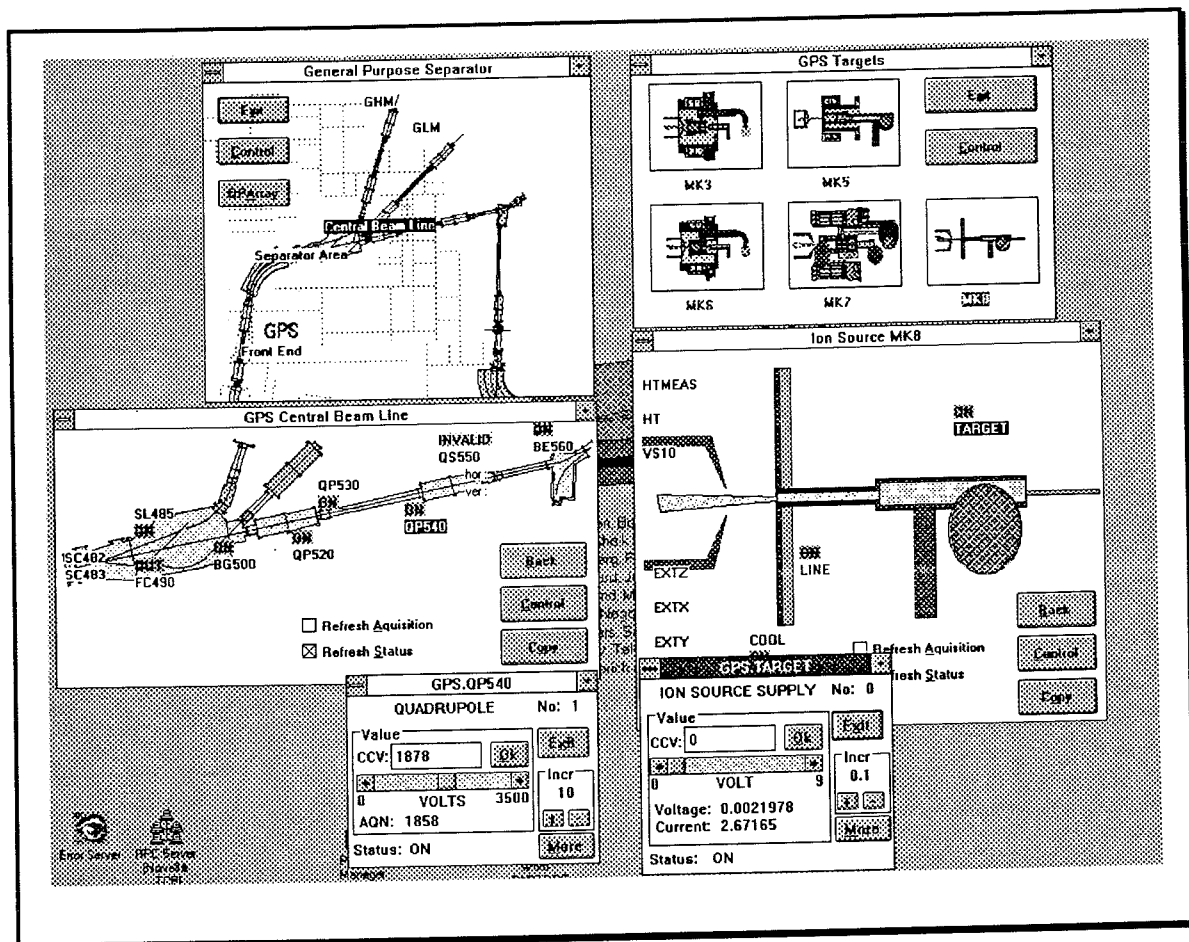


Figure 5.3. On two vertical columns are two applications with the same internal structure.

In figure 5.3. we have two examples from two completely different applications, that shows how internal consistency was understood. The first example is taken from a synoptic that controls the group of elements located on that trunk of the beam transport system. The second example shows the control of a target where the 1 GeV proton beam hit the target and the heavy ions are generated. The type of commands, the size of buttons, the text, the type style, the relative position, the procedures and the colors are kept the same. In particular when an application is called always shows an

overview of the hardware you are able to control. When you click the part of the hardware you want to control a new window shows a zoom of that area with the names of all elements you are able to control. By choosing one element, or by double-clicking one element you'll ask the control panel (or knob) of the desired element. For each type of element a different control panel is used, and a database contains the description of the control panel for each element. For what regards screen layout we decided to standardize the screen layout as shown in Figure 5.4.

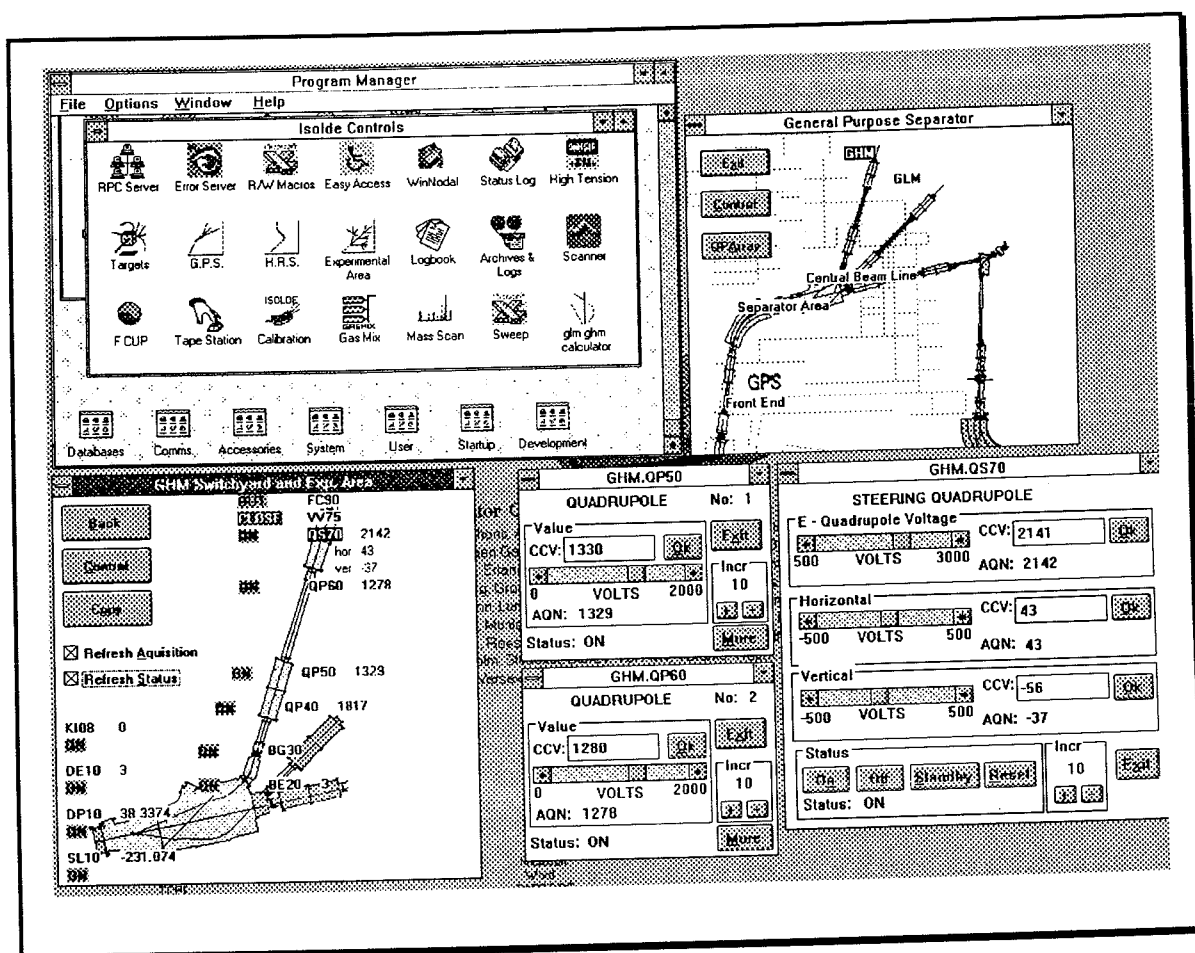


Figure 5.4. Typical screen representation, with the screen divided into four main areas.

The screen is divided in 4 areas: The top left one is reserved for the Program Manager of Windows. In this way the user is at any time able to call new applications, utilities, change settings and printers, access databases and so on. On the top right we

thought to display the first window of the application, usually a synoptic. From this first window the operator is be able to call other windows which are displayed on the bottom left of the screen while the bottom right is reserved for the control panels of the elements.

Navigability is the last concept in organizing. To improve navigability we added priorities within each application. For example windows which should not receive commands, are always disabled and the same happens to buttons that should not be clicked. Buttons also shows their inoperativity by assuming a light gray color , as standard in Windows.

5.3.3. Economization.

This principle can be split into four main concepts, which are: simplicity, clarity, distinctiveness, and emphasis.

Simplicity reminded us not to overload of information the control panels. We decided to add the possibility to show or not the status and/or the working values of the elements on the synoptic, as shown in Figure 5.5.

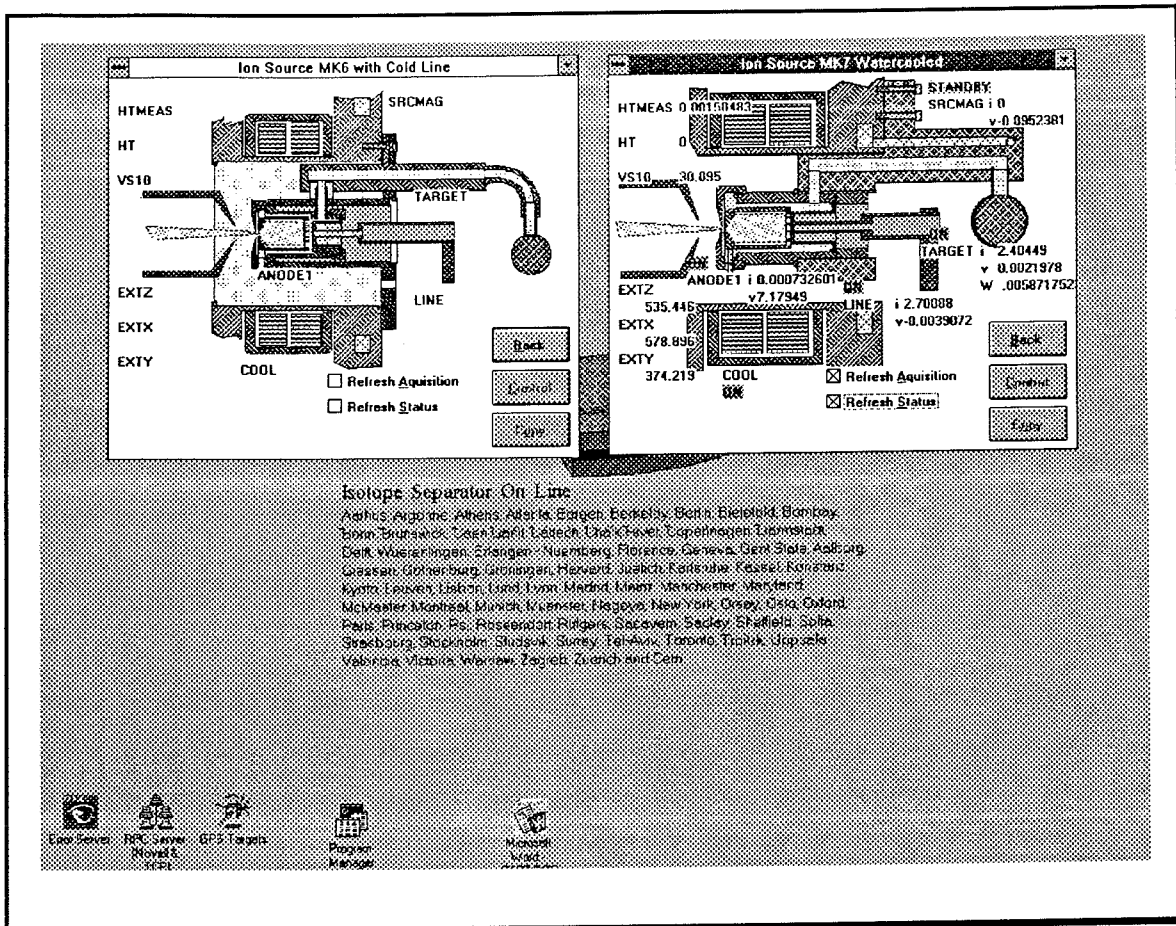


Figure 5.5. One synoptic is clearer but without information. The other one is more confused but shows the whole status of the target.

We then decided to implement a centralized service with the duty of creating knobs when required. This presents some advantages. The use of a knob permits to have detailed information only about the elements the operator is interested in. Moreover it is possible, in this way, to know if two applications, running on the same console, are demanding the same knob. The rule is then to create only one knob which will serve both the applications. This permits to avoid redundancy and confusion.

Clarity is the second way to be economical. We thought to obtain clarity by being consistent in the implementation of synoptics, icons, and other application. As an example the FCUP application, was suggested from a group of physicists that would like to tune the beam with the help of an indicator that represent the intensity of the

beam. The first solution was a simple digital representation of the intensity of the beam. But numbers requires interpretation which needs time and leads to errors. We then built an application that showed the intensity in an analogic way, as shown in figure 5.6.

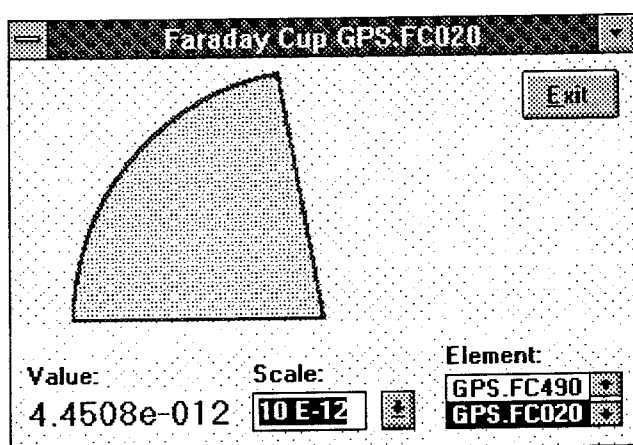


Figure 5.6. The Faraday Cup Application.

The slice is proportional with the intensity of the beam and the external border of the slice turns to green or red depending on an increase or decrease of the intensity.

The windows created by the applications follow rules and keep colors from the standard of Windows. We thought, in fact, that distinctiveness and emphasis proposed were good enough for our applications, and we avoided, in this way, to double a standard.

5.3.4. Communication.

Factors that affect communication are slightly less self explaining and are: legibility, readability, typography and symbolism.

Legibility, readability and typography concern mainly the text in the applications. Our graphical environment do not use large amount of text. Considerations were made only on the use of the same type and size of fonts in all applications. Fonts type and size were chosen in relation with the dimensions and resolution of the screens. Dark fonts on bright backgrounds has been chosen considering that the control room is a brightly lighted ambient. Moreover text flush from left and numbers from right, uppercase and lowercase characters have been used to speed up reading.

Symbolism is a complex concept which includes consistency and imagery. The scope to convey information can be achieved using photographically real, abstract or mixed images. The problem in this case that visual identity changes from person to person. A physicist who knows ISOLDE won't have any problem in understanding a synoptic which we used for our control system. An external person in seeing that synoptic may equally think that the total size of the machine is one meter, or a few hundred meters. In our GUI we tried to use the photographically real imagery where possible as shown in Figure 5.7, where we show the vacuum system, the target and the gas mix synoptics with relative icons.

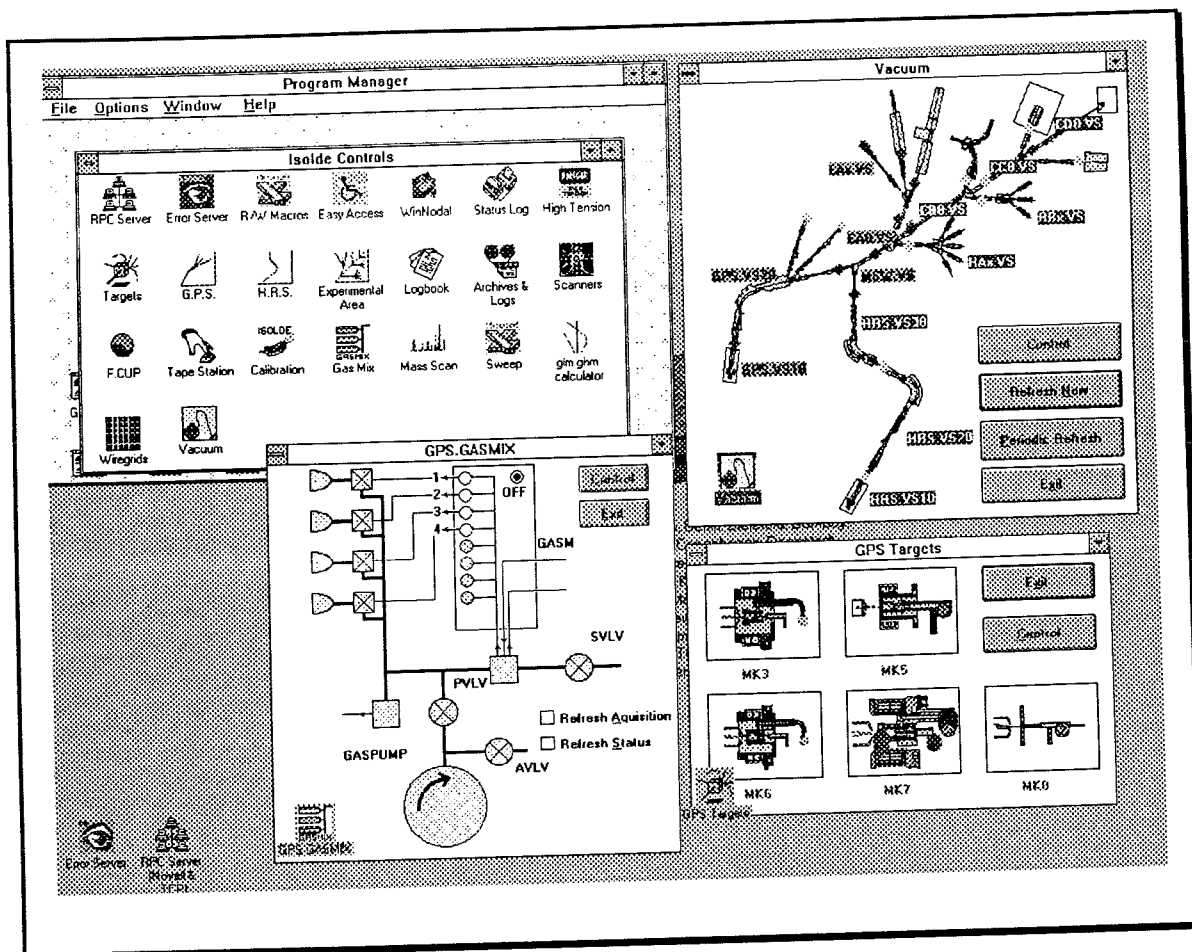


Figure 5.7. Photographically real imaginary has been used for synoptics, GPS gasmix and GPS targets.

Abstract imagery was chosen for applications such as Mass Control, Wiregrid, Faraday Cup and GLM/GHM Calculator as shown in figure 5.8.

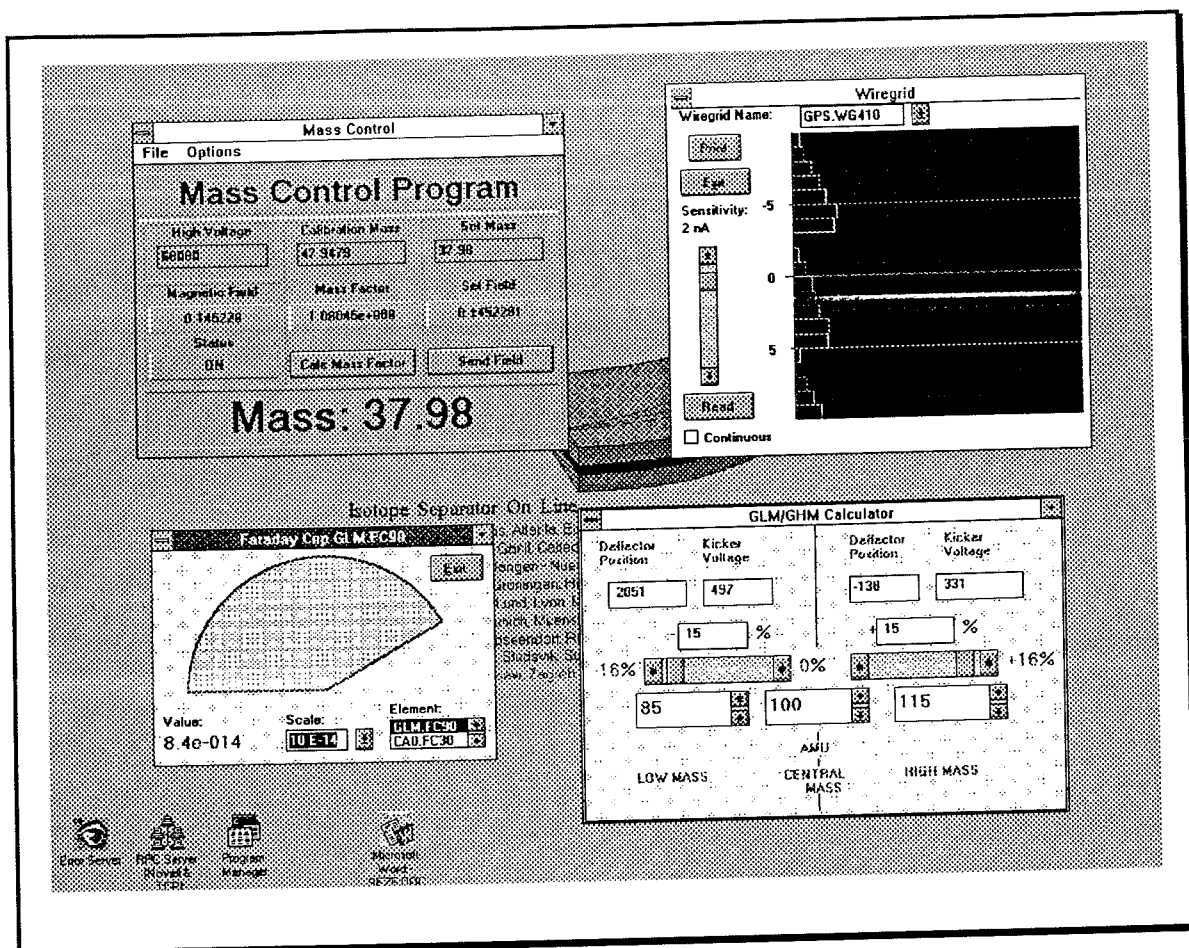


Figure 5.8. Abstract imagery was chosen for applications such as Mass Control, Wiregrid, Faraday Cup and GLM/GHM Calculator.

5.3.5. Multiple Views.

Great attention has been paid on the concept of multiple views. Three points, in particular, have been considered.

The first one regards the type of representation. Where possible we tried to double the type of representation. We always tried to have an analogic representation and a digital one. This to permit to convey a qualitative as well as a quantitative perception of the interested value. Scrollbars and pies have been used to give visual representation of values while standard digits were always present to give the exact value and to be stored and retrieved.

The second point threatens the problem of showing information. The screen is the main but not the only output of the console. The output can be easily redirected to printers when, for example, some data or configurations need to be saved on paper. The point is that users will likely use b&w laser printers to do hard copies of the screens. Color printers were not considered because of related speed and cost. The b&w hard copy of the screen must contain the same information of the screen if we want to consider it as a valid alternative. We were then not allowed to concentrate information on colors only. Colors should just be considered as a help to convey information. For this reason all controls that make use of colors always double their information with text. In practice, text is the vehicle of the information, color helps the system to convey meaning to the user.

The third point is about links across applications and metadata features. In the implementation of the control system is important to have the possibility to take data from one application and paste it into another application or into a document. Windows has a built in, and very powerful Dynamic Data Exchange (DDE) mechanism. This permits everyone to define either static or dynamic links within different applications that supports DDE.

Excel supports DDE and this is one of the reason for which we choose it as our database. Moreover on some applications (such as synoptics) we added a feature that permits to store one variable of the system on the clipboard, so that it can be pasted in any applications that supports DDE (Winword, Excel, Paintbrush. etc.). This link can be either static or dynamic, in the sense that the link can be automatically refreshed every time the value of the variable changes. In figure 5.9. we see an example on the possibility offered by a link between the control system and WinwordTM.

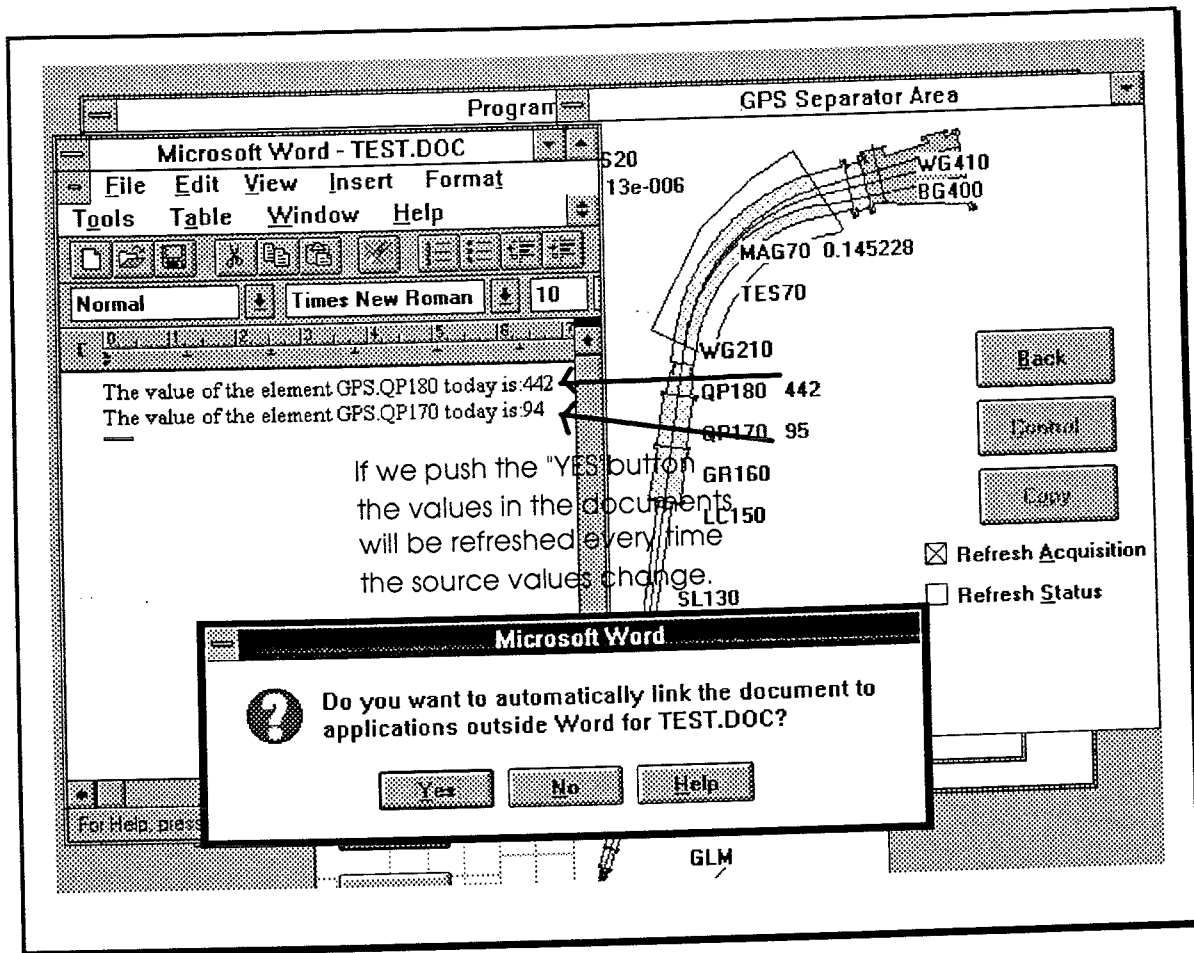


Figure 5.9. When we try to open the Word file TEST.DOC we are asked if we want the linked values to be refreshed if the source changes.

This introduces the metadata concept. The Windows environment proposes, with DDE, a very powerful feature that checks and converts file formats and permits to exchange data, text and graphics between applications. We decided to support this feature in our system, and as a consequence, all commercial software used in the system supports DDE.

5.3.6. Color.

The discussion about colors will be slightly more complicated, due to the complexity of the tool. The articulation of the discussion will affect principles like color organization, economy, emphasis and communication.

Recommendations about color organization suggested to use color to group related items, and to use a consistent color code among the applications. We interpreted those guidelines and decided, for example, to use the same background color in all windows of one application and to use different background colors for different types of applications Fig. 5.10.

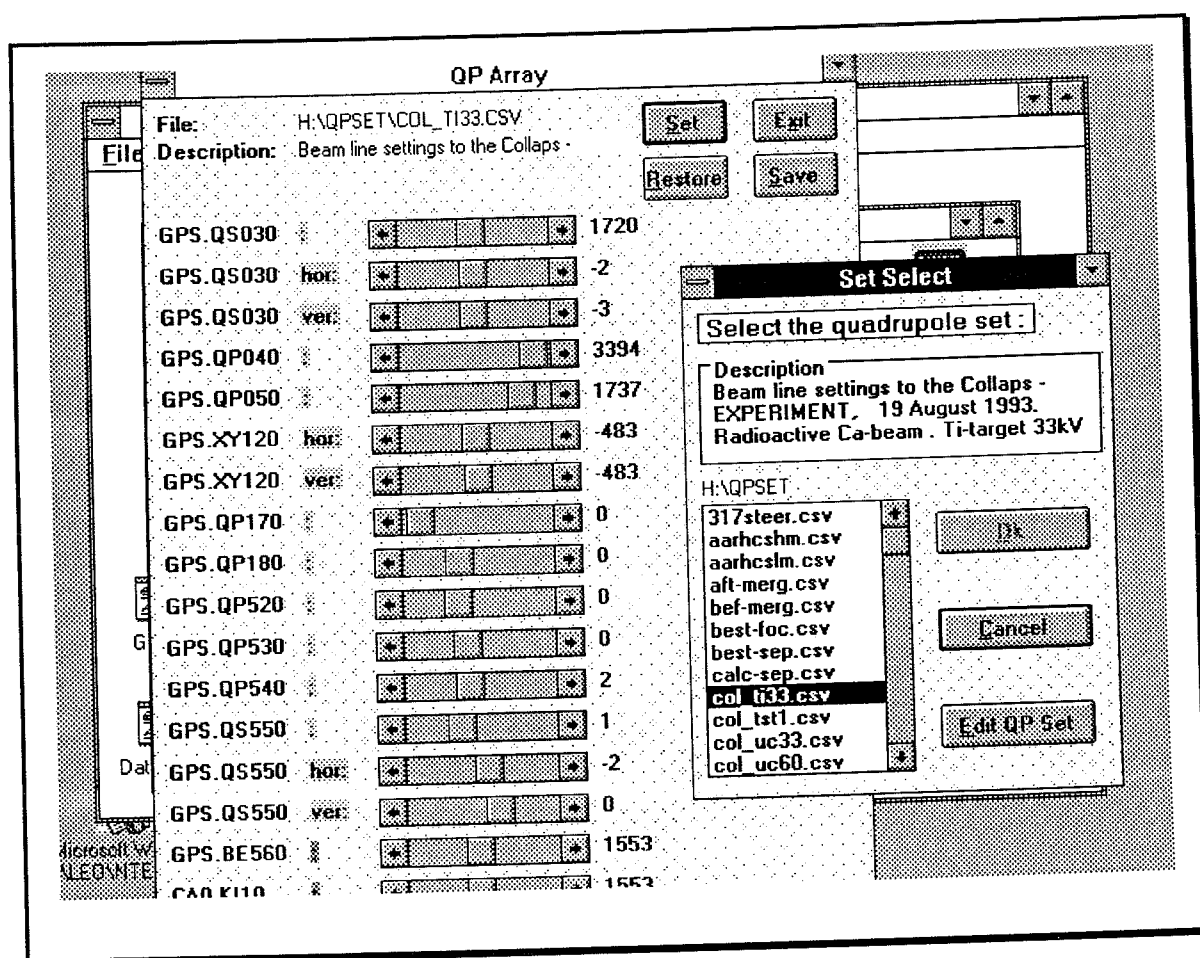


Figure 5.10. The set select window of the QP-Array application permits to choose the set of elements to control. The QP-Array windows permits to modify, store and retrieve configuration values.

Moreover as long as we decided to use a standard set of terms (as OPEN/CLOSED, ON/OFF, IN/OUT, STANDBY, INVALID) we decided to code a color language and to relate it with those terms. In particular we decided to relate the green color with the situations that permit the beam to reach the experiment. This

happens when valves are OPEN, power supplies are ON and intensity measurement instrumentation (Faraday Cups) are OUT. We decided to use black text on green background. Red was chosen to show circumstances that stop the beam. This is related with valves that are CLOSED, power supplies that are OFF, and faraday cups that are IN. To have a better visibility we chose white text on red background. a yellow background with black text was chosen to show situations which may stop the beam, such as power supplies in STANDBY or INVALID configuration, or communication problems between the console and the hardware. A example of this rule is shown in figure 5.11.

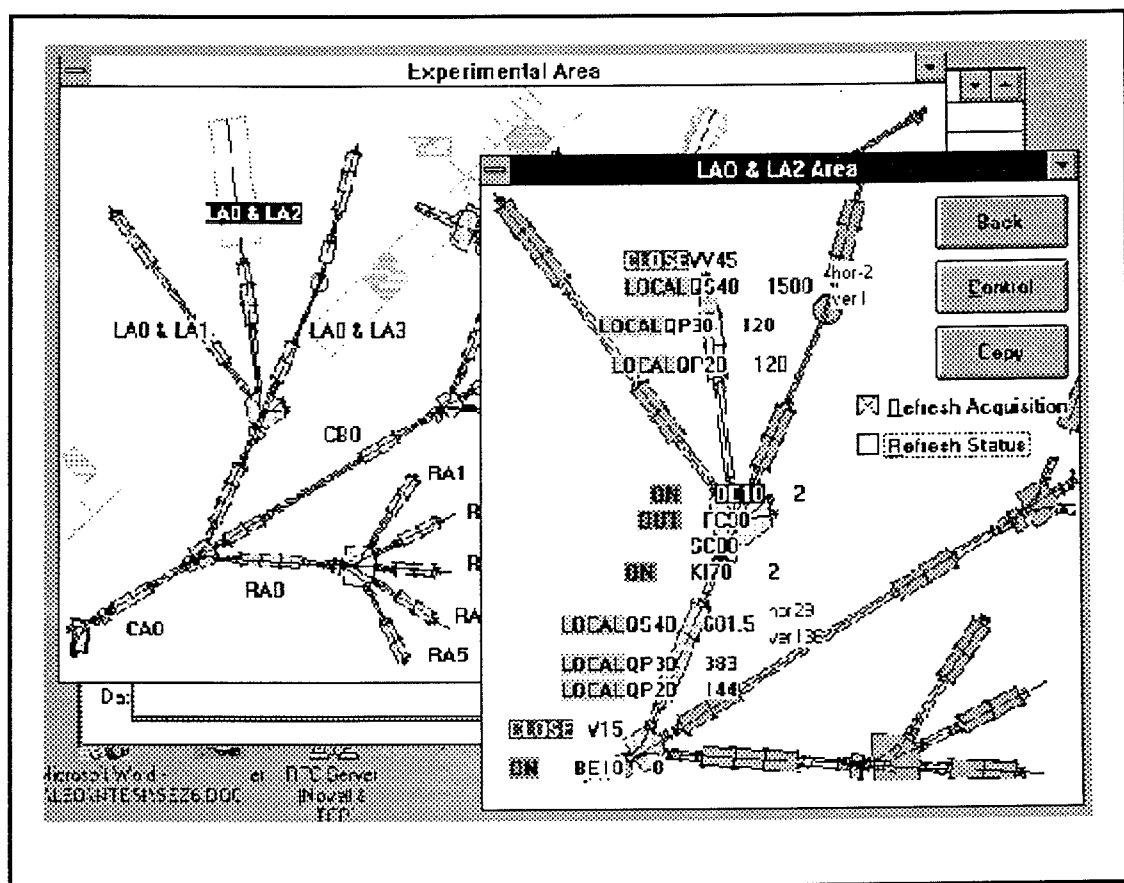


Figure 5.11. Although the information relay on text the color code help to convey information.

We also decided to use the color palette standard in Windows so that it is clear which window has the focus (in Windows the title bar of the focused window is blue while all the others are white) and the user feels well with the system.

The decision to group many terms into one color was taken considering recommendations about color economy. Those suggest that a user is able to remember the meaning of 5 ± 2 colors. Infact color should be used as a redundancy, and to help convey meaning. They should not be used as holders of information. They should transmit the concept desumed from the information just to discharge the user doing that. Under this point of view, grouping many terms into one color is not a loss of information but an advantage for the user. The text inside always give the information which is also clear if b&w hard copies of the screen are generated (like the figures we are proposing).

Color emphasis principles helped us in the use of the color in the sense that red, yellow and green in the order focus the user attention. The user attention infact should be focused on the, hopely, few things that are going wrong, rather than on the whole which is working fine. In a competition for user attention, with light background, yellow wins on green, and red on both. High chroma red, aid then faster response.

To avoid confusion we tried to use those colors only in the described situations and for all the remaining we used low chroma grays, blue and yellow that are at the bottom of the spectral sequence and are perceived as little attention demanding colors.

In our guidelines color communication said that the outer edges of the retina are not particularly sensitive to colors, and suggested high chroma colors to be used only in the center of the screen, and to use blue black white and yellow near the periphery. The problem in our case was that, for example, we wanted an icon on the low left area of the screen with an application associated that checked the communications between consoles and hardware. If there were problems this icon should show the operator the presence of the problem. We wanted to keep also for that icon the standard color code, green if everything is OK and red if there are problems. But red does not attract the operator's attention if situated on the border of the screen. We opted then for a mixed

solution, The icon, in case of problem should have blinked few times, while changing from green to red. In this way the blinking was conceived to attract the viewer's attention and the red to convey the meaning.

Another case in which recommendations for color communications were useful is in the synoptics where we used black to write the most important information, such as the name of the elements, and blue for data related to each element (slightly less visible).

5.4. How applications are implemented.

5.4.1. Application structure.

The applications I did are all written in Visual BasicTM, and are: GPS, HRS, Experimental Area, QPArray Targets, F.Cup, GasMix, and Vacuum. Their software structure is somehow similar. For this reason we will only see an example on how an application has been implemented, taking into account that all the others are not very different. The application we are going to analyze is the **General Purpose Separator (GPS)**.

The application consists in seven blocks, as we can see in figure 5.12 in the **GPS.MAK** list.

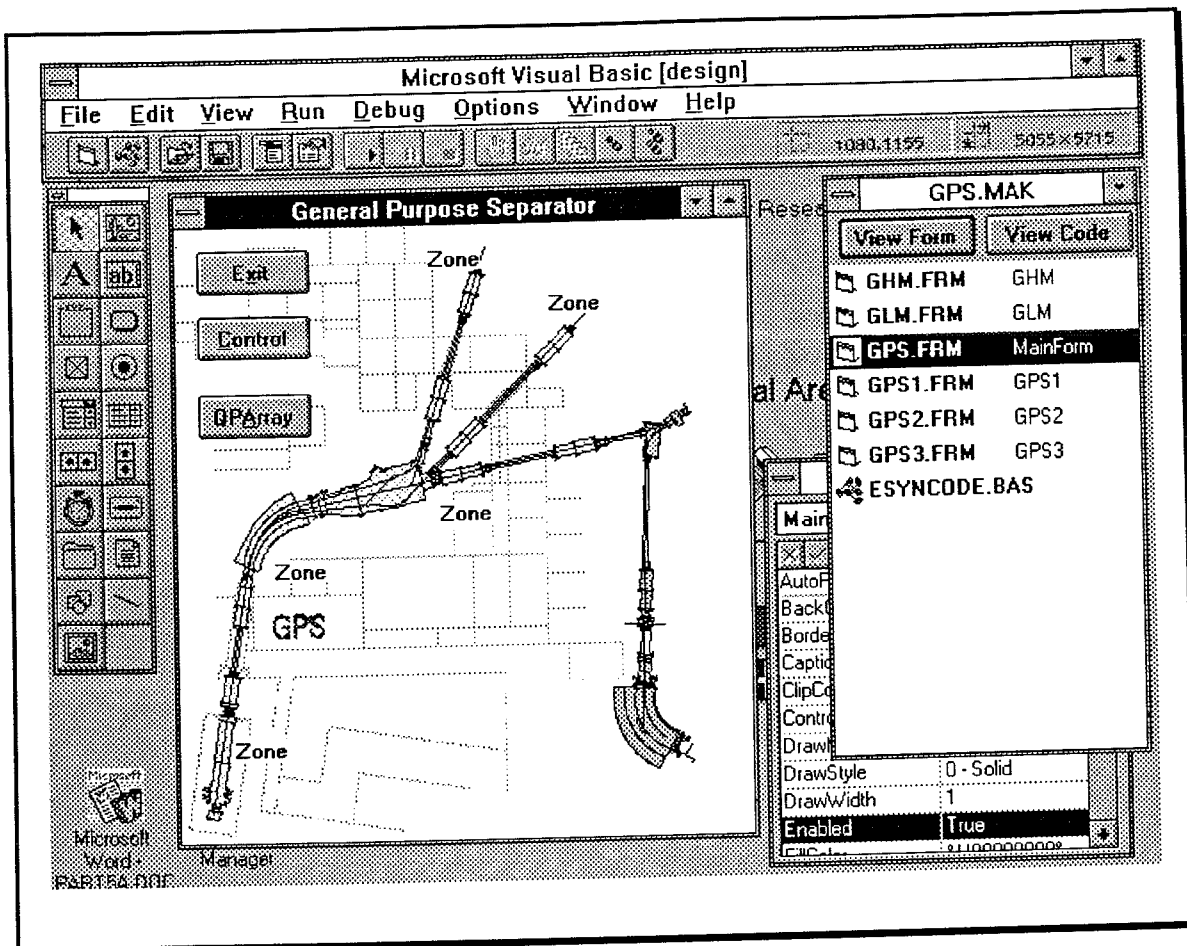


Figure 5.12. The Visual Basic™ development environment, with the view of the main form and the project list.

Six of them are forms, while the last one is a module. A form consists in a window on the screen, usually containing drawings, labels, buttons, scrollbars, and so on. Attached to each of these elements, there is the code that is executed if a required operation is done on the element. Visual Basic™ is an event-driven environment. This means that when a program is executed, its process is an idle loop scanning for predefined events. When an event is generated, as an example a button is clicked with the mouse or a label is double-clicked, then the code attached to the event is executed. The forms can be visible or not. Only used forms are made visible, to limit screen competition and confusion. Only code contained in visible forms can be executed.

Modules are groups of functions that can be called and executed by all forms. Error handling functions, writing to hardware and reading functions and some others are contained in the **ESYNCODE.BAS** module.

We will now see how the **MainForm** form is conceived and one of the others, as the rest is quite the same.

5.4.2 The MainForm form.

This form is the first form that is made visible when the program is called and is the one shown in figure 5.12. The goal of this form is to show the area that can be controlled with this application, and to permit the operator to ask for the other forms that have control possibilities. The background picture is a bitmap drawing obtained by importing in DesignerTM using the dxf format of the AutocadTM layout of the project, by resizing it with DesignerTM, and finally by touching and coloring the picture with PaintbrushTM. Referring at the form we can say that there is the possibility, by clicking the **QPArray** button, to call an application that permits to store and retrieve configurations of elements with all their properties. The **Exit** button quits the program, while the **Control** one opens one of the other forms showing the selected area. If no areas have been selected the **Control** button remains of a light gray color and is not active. To select an area, the user should click with the mouse on the name of the area he wants to select.

Let's now see how all this can be done. To get used with the working principle of an event driven environment, we will see how the **QPArray** button works. If the user wants to open the **QPArray** application he has just to click on the relative button. Every command has a number of properties and one of them is the identification name. The identification name of the **QPArray** button is **QPCommand**, and if you click on that button, the subroutine **QPCommand_Click** is executed. In figure 5.13 this subroutine is listed.

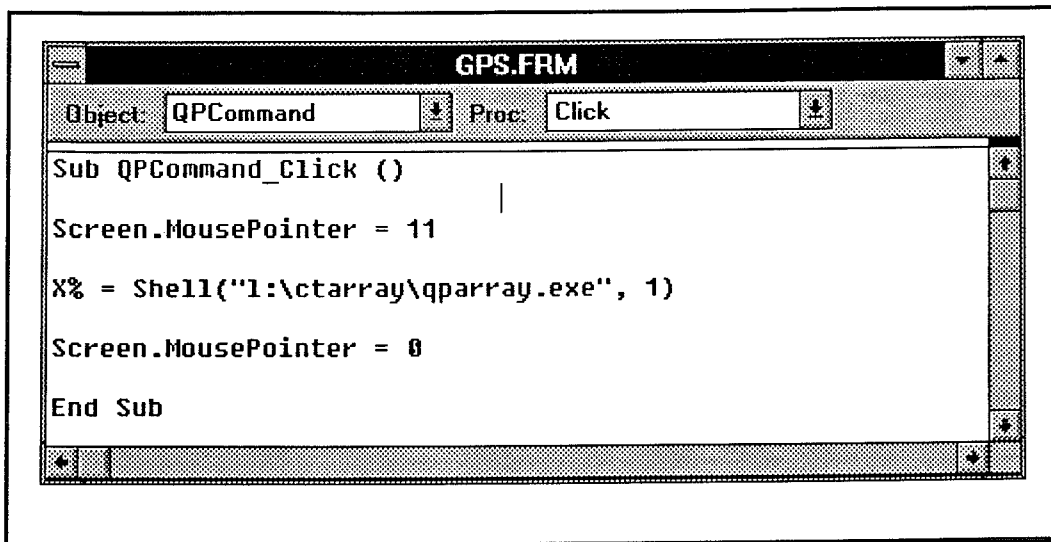


Figure 5.13. The QPCommand_Click subroutine.

What happens is that the mouse pointer assume the shape of a hour-glass, the program **QPARRAY.EXE** is loaded and the mouse pointer assume back the default shape.

Let's now see the other subroutines. The **Exit** button subroutine is not relevant. What is relevant is the **Form_Load** subroutine that is executed when the form is loaded. Figure 5.14 contains this subroutine.

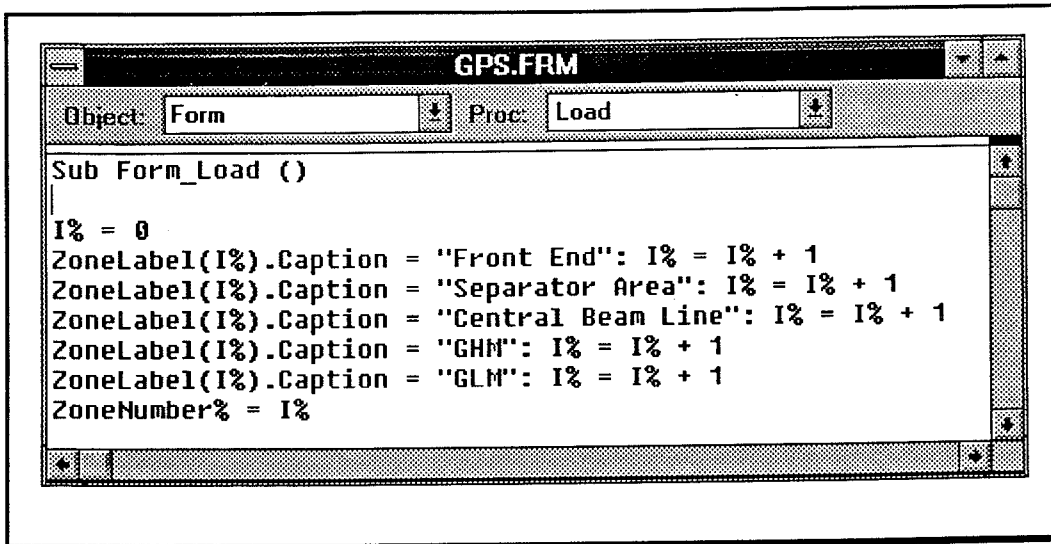


Figure 5.14. The Form_Load subroutine.

To understand this code we need to know that **ZoneLabel** is an array of labels, and that **Caption** is a property of **ZoneLabel** that contains the text of the label. When the subroutine is executed, it loads the names of the areas in the labels that are marked with **Zone** in figure 5.12, and initialize the two variables **ZoneNumber%** and **ZoneSelected%** to the number of existing areas and -1 (**FALSE**) respectively.

Now that the form is loaded and initialized, the operator is able to ask for a zoom of one area that has control possibilities. Imagine that the operator wants to open the **GLM** area, which is the last area and consequently has an index number of 4. The operator clicks the label with index number 4. The **ZoneLabel_Click** subroutine, listed in figure 5.15 is executed.

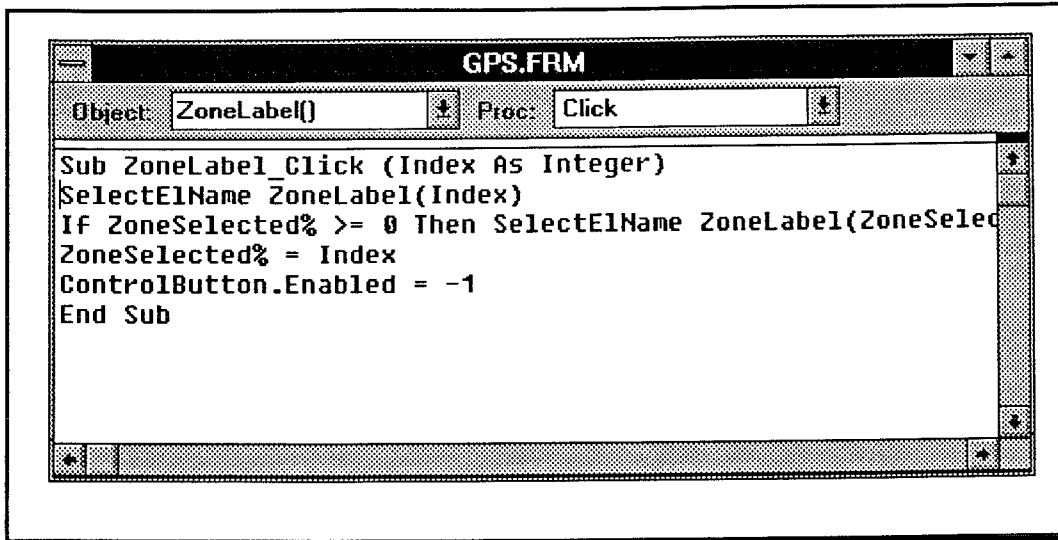


Figure 5.15. The ZoneLabel_Click subroutine.

When this subroutine is called, first of all the **SelectElName** function, contained in **ESYNCODE.BAS**, with **ZoneLabel(Index)** as parameter is executed. This function just exchange the **Backcolor** property of the label with the **Forecolor** (the color of the text) one, and is introduced to mark the selected zone.

The next statement means that the same function has to be called if **ZoneSelected%** is not false, that means that if other areas were selected before, a second exchange take them back to the normal colors. In the next step the actual area is registered as selected and, as we are now sure that an area is selected the **Control** button is enabled.

The operator that selected the area is consequently enabled to push the **Control** button, with the consequence that the attached code is executed (Fig. 5.16).

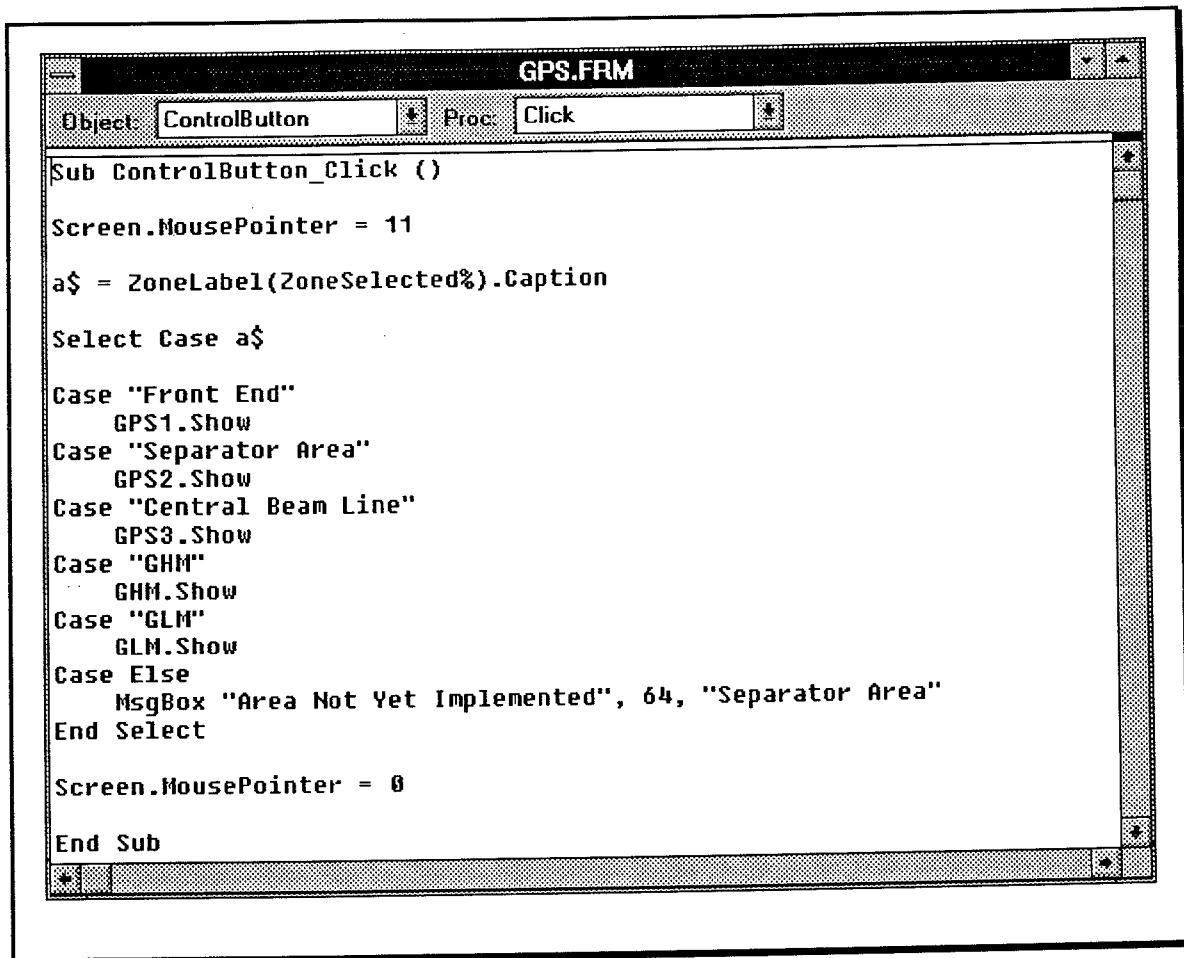


Figure 5.16. The ControlButton_Click function.

This subroutine begins with changing the mouse pointer to a hour-glass, and after having loaded the **Caption** property (which is the text of the label) in the **a\$** local variable, it test this variable with the possible names and, if successful it make visible the requested form. This is not the case, but in some other applications, not all the areas were implemented. This is why an **ELSE** case has been introduced, with the effect of displaying a dialog box in case of unsuccess. Once the new form is loaded and displayed, the mouse cursor change back to its default shape.

5.4.3. The GLM form.

The new form is slightly more complicated, as uses the Dynamic Data Exchange (DDE) possibilities offered by the environment, and it is necessary to cope with error handling situations.

Figure 5.17 shows the new form and the **Form_Load** subroutine.

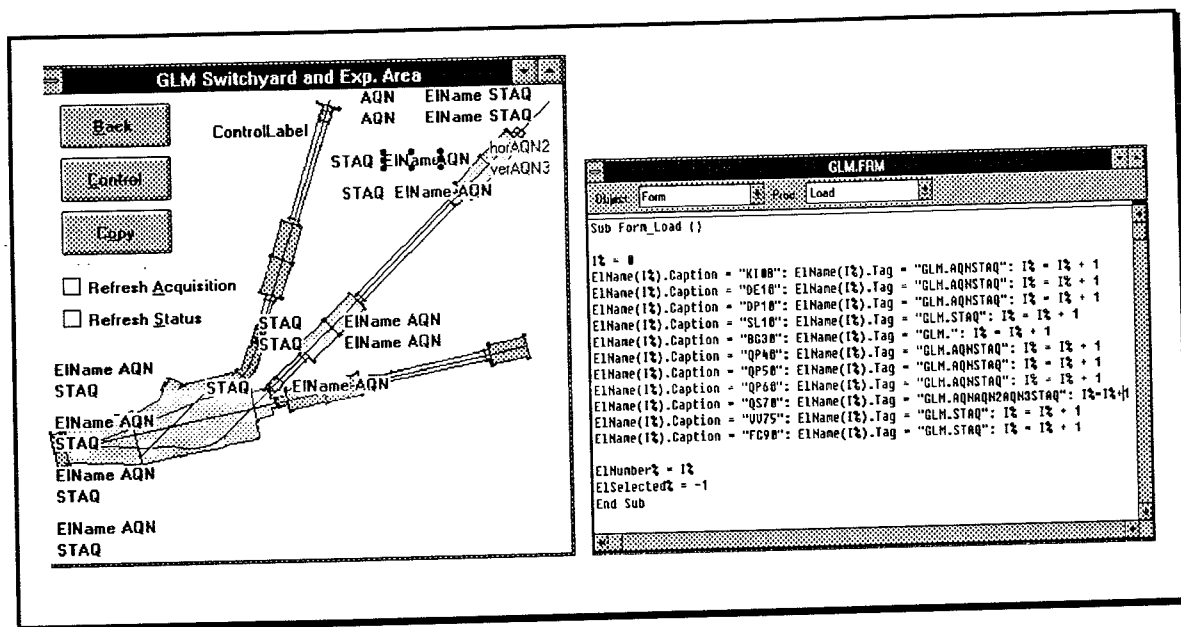


Figure 5.17. The GLM form and the **Form_Load** subroutine.

This form is similar to the prior one for what regards the background picture, the labels that here are grouped into 7 array instead of one, and the **Form_load** subroutine. It is different for what regard the functions, and we will now see how. First of all let's look at the **Back** and **Copy** buttons. The first one hides the current form while the second one permits to copy the acquisition value of the selected element into the clipboard of the Windows system. This is useful in order to establish links between applications, as for example this application and ExcelTM or WinwordTM.

Suppose now that the operator wants to have the control of an element, let's say the **GLM.QP40** quadrupole. Then he has to select in the usual way the element label and push the **Control** button.

This time the code attached to the button is different and is listed in figure 5.18.

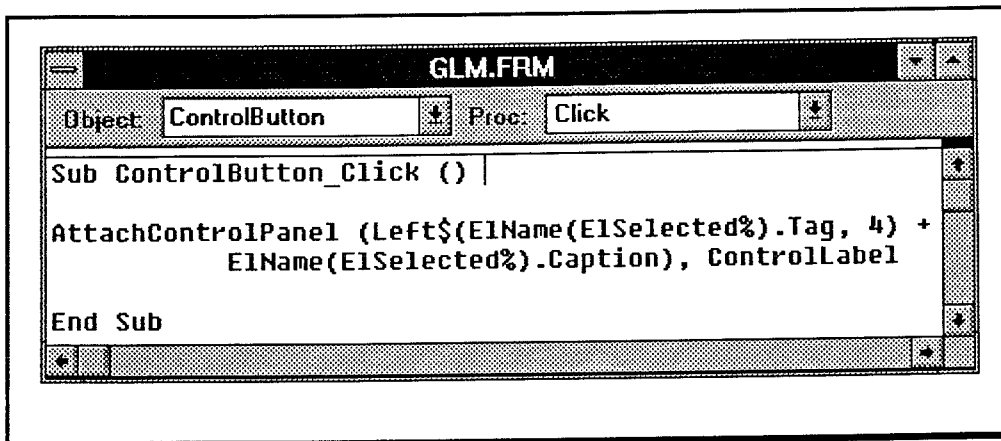
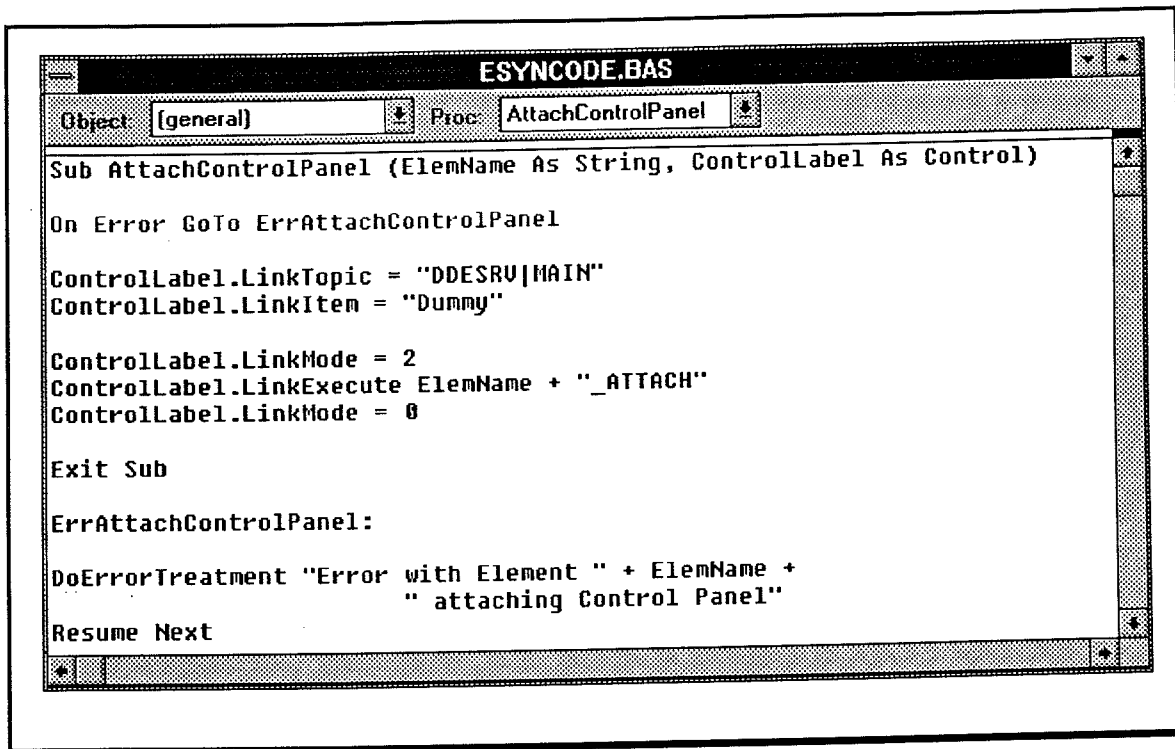


Figure 5.18. The ControlButton_Click subroutine.

This code is only a call for a function which is listed in figure 5.19., but it is interesting to see the parameters that are passed.

Infact those parameters are a string containing the first four characters of the **Tag** property of the selected label, which is a sort of hidden text attached to every label, plus the **Caption** property of the label, and a **ControlLabel** which is used as hidden buffer.



5.19. The AttachControlPanel function.

This function begins with a statement that makes the process jump to directly to the **DoErrorTreatment** function if errors occur. In this case after the error treatment the **Resume Next** command reestablish the situation. After it, the properties of the **ControlLabel** label are used to establish a DDE link with the **Server** application. The scope is to ask to the server application to open the control panel of the selected element. The procedure is as follow:

The **LinkTopic** property determines the server application name and the subject of a conversation in a DDE link. In our case **DDESRV** is the name of the application and **MAIN** is the fundamental data grouping used in that application.

The **LinkItem** property specifies the data passed to a client control in a DDE conversation. In this case no data are passed and we gave a **Dummy** value to this property.

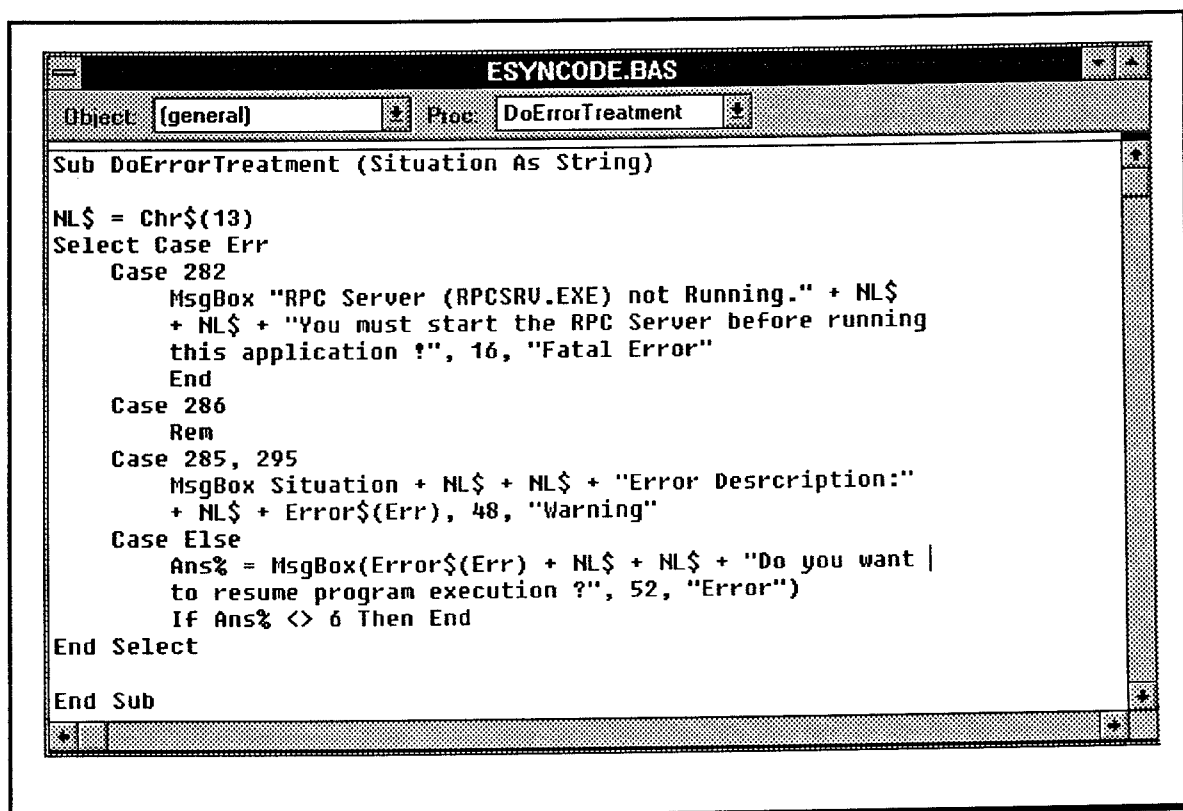
The **LinkMode** property determined the type of link used for a DDE conversation and the meaning of the possible value are: 0 for no DDE interaction, 1 for a hot link

(that means that the client control is updated each time the linked data changes), and 2 for a cold link (that means that the link is never refreshed). In our case a cold link is required.

LinkExecute is not a property but a method, and its action is to send a command string to the other application in a DDE conversation. In our case we send the element name and the command "**_ATTACH**" that will be interpreted by the server application.

Finally to stop the conversation a **LinkMode = 0** is sent.

The error treatment function is listed in figure 5.20. and consists mainly in a **CASE** structure to test the error code and make the correct information window describing the error.



```
Sub DoErrorTreatment (Situation As String)
NL$ = Chr$(13)
Select Case Err
Case 282
    MsgBox "RPC Server (RPCSRV.EXE) not Running." + NL$
    + NL$ + "You must start the RPC Server before running
    this application !", 16, "Fatal Error"
End
Case 286
    Rem
Case 285, 295
    MsgBox Situation + NL$ + NL$ + "Error Description:"
    + NL$ + Error$(Err), 48, "Warning"
Case Else
    Ans% = MsgBox(Error$(Err) + NL$ + NL$ + "Do you want |
    to resume program execution ?", 52, "Error")
    If Ans% <> 6 Then End
End Select
End Sub
```

Figure 5.20. The DoErrorTreatment function.

Other possibilities offered by the form are to visualize on the picture the status of the elements and the values of the tension. This selection is done by selecting the options **Refresh Acquisition** and **Refresh Status**. The process is similar to the one we saw for the control panel, with the difference that, this time, a hot link is established, so that each time a value change the information is refreshed.

6. The implementation of the ISOLDE control system: the Front End Level.

6.1. Introduction.

The front end computer is the connection between the equipment and the control system. The advantages guaranteed by the possibility of local interactivity, for testing purposes, pushed for the use of PCs also at this level. Moreover performance test showed that 80286 PCs have enough CPU power to drive the equipment (Ref. 26). Another advantage, using this family of front end computer, is the possibility of local mass storage, useful when debugging or testing the equipment. This architecture supports three kind of control boards. These are:

- CAMAC, to allow the recuperation of all the existing hardware of the old project;
- GPIB to drive sophisticated instrumentation;
- Industry Standard Architecture (ISA), by the use of PC/AT boards fitted into the PC directly or in extension crates.

Depending on the type of equipment to control adequate control boards had to be chosen. The elements I had to control were : quadrupoles, steering quadrupoles and deflectors. In this section we are going to describe how a quadrupole works and the basic principles of its control, taking in mind that a steering quadrupole and a deflector are quite similar. A quadrupole will from now on be called QP, and thus quadrupoles, QPs.

6.2. Description of a QP.

The element we chose for this analysis is the QP (Fig. 6.1.) because it is simple, being driven directly from the FEC via some ISA boards, and complete enough to show the main features of the control system.

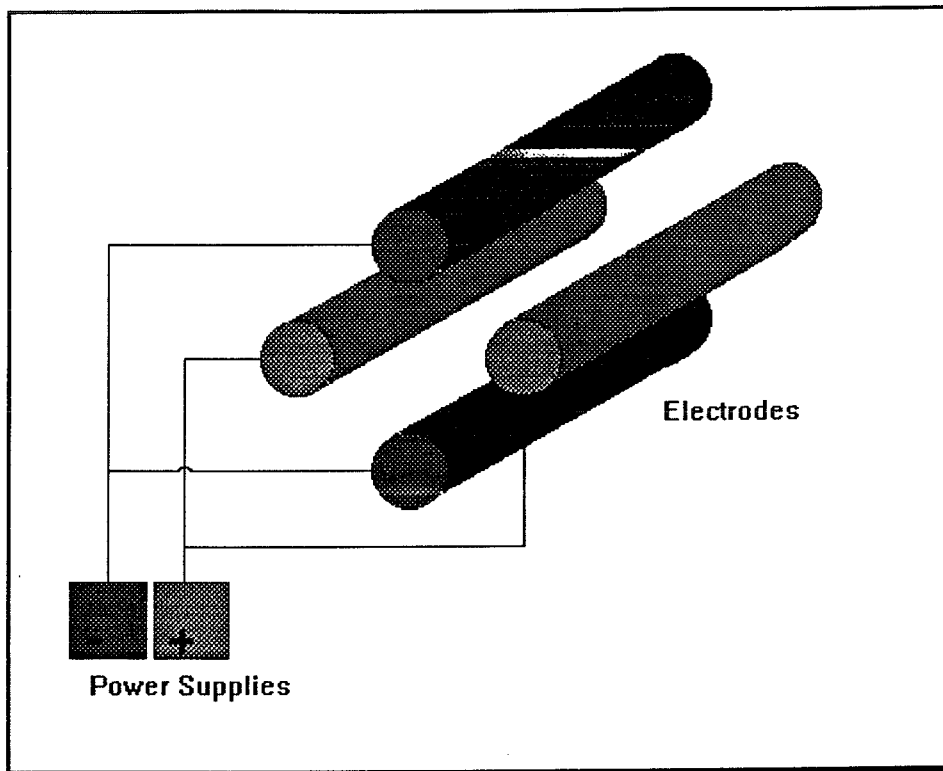


Figure 6.1. Schematic of a QP.

The purpose of a QP is to focalize the beam. The focalization is necessary because all the particles of the beam have the same type of charge and tend so to drift. In order to contrast this effect a particular electric field is needed. A QP is the element where this particular field is created. We know that the force that deviate a charged particle is proportional to the present electric field. To obtain a focalization we need a field with the characteristic of having its intensity proportional with the distance from the axis of the element. This can be achieved using an electrostatic lens configured as a QP. Physically a QP is formed by four cylindrical electrodes, disposed on the edges of a square, and each connected to power supplies.

The theory that describes the QP lenses considers hyperbolically shaped electrodes. A good approximation is possible if (see Figure 6.2.) R is equal at $1.147 G_0$.

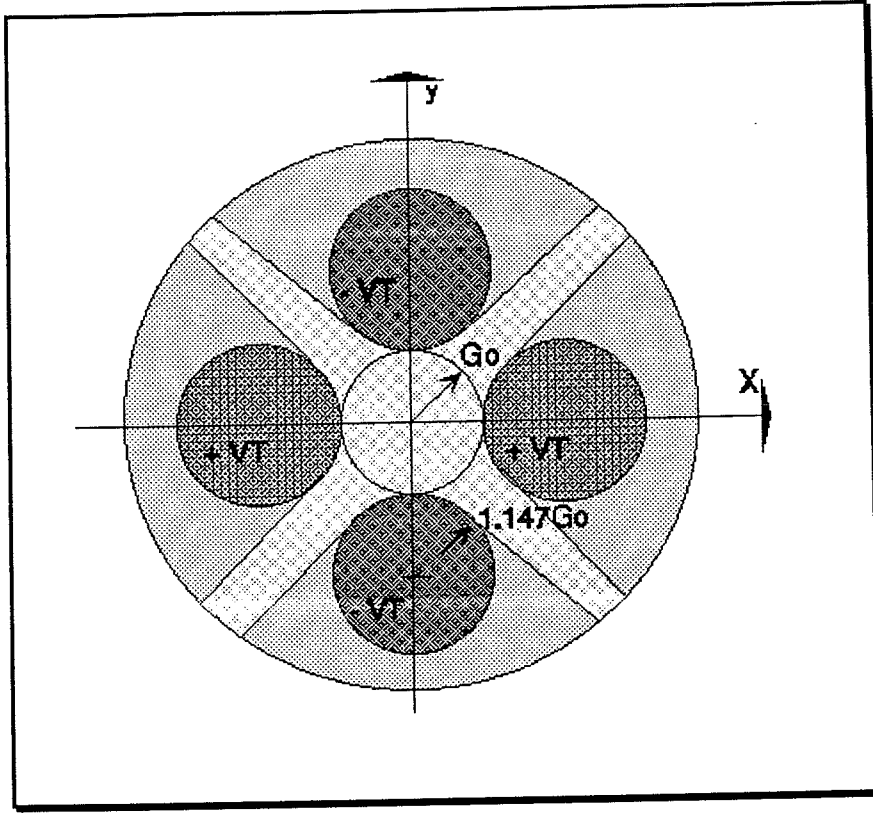


Figure 6.2. Approximation of hyperbolic electrodes with circular ones.

For an electrostatic QP with a diameter of $2G_0$ and electrode potentials $\pm V_T$, the electrostatic potential distribution relative to the optical axis is (Ref. 27):

$$V(x,y) = [(x^2 - y^2)/G_0^2] V_T$$

With $\mathbf{E} = -\text{grad } V$, the components of an electrostatic field strength $\mathbf{E}$ are then obtained as:

$$E_x(x,y) = -\delta V / \delta x = -(2V_T/G_0^2) x$$

$$E_y(x,y) = -\delta V / \delta y = (2V_T/G_0^2) y$$

$$E_z(x,y) = -\delta V / \delta z = 0$$

yielding constant field strength gradients along the x and y axes. We can then assume that the equations of the potential inside a generic QP is

$$E_x = -A \cdot x$$

$$E_y = B \cdot y$$

Coefficients A and B are responsible for a force which is proportional to the distance of the particle from the optical axis of the lens. As far as particles are charged positive, the negative sign in front of the A coefficient means that the beam will be focused on the X plane. The opposite happens for the Y plane in which the beam will be defocused. For this reason, to obtain a correct focusing, it is always necessary to use two QPs rotated 90 degrees one from the other. This configuration is known as a doublet and has been extensively applied in the ISOLDE optics. It can be useful to specify here, that the total effect of a defocusing and focusing or a focusing and defocusing action, results in a positive focusing. The explanation is that the focusing effect take place far from the optical axis of the lens while the defocusing effect happens near to it. Because of the linear growth of the focusing or defocusing effect from the central axis towards the periphery, the focusing effect is always stronger than the defocusing one and the total effect is a focusing action, as shown in Figure 6.3.

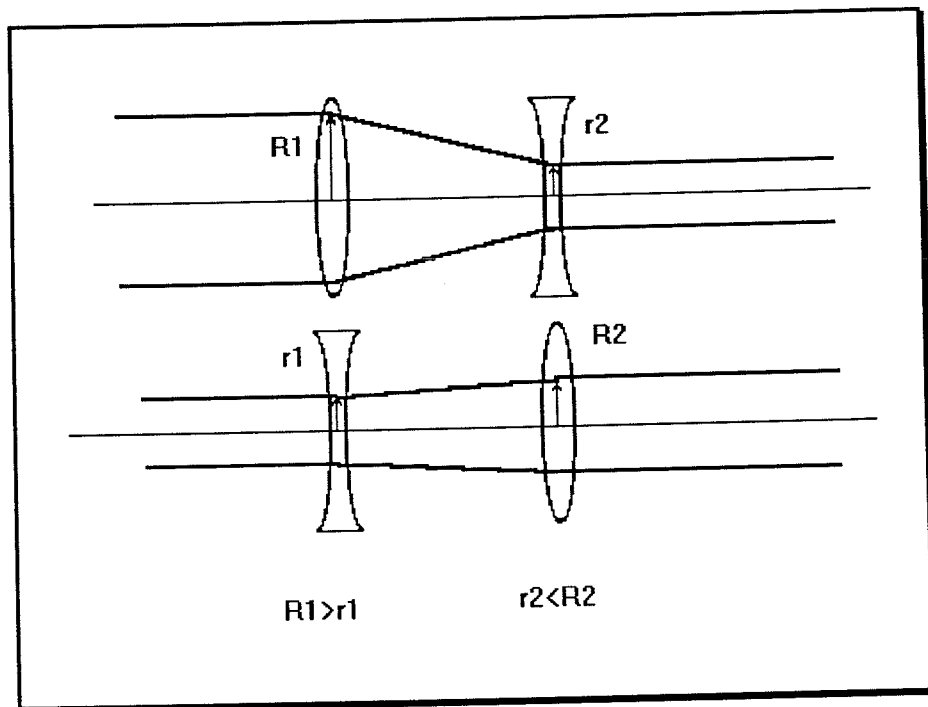


Figure 6.3. Explanation of the focusing/defocusing effects. The focusing effect is always stronger because $R1$ is bigger than $r1$ and $R2$ is bigger than $r2$.

Now that we know the behavior of a QP we need to decide how to control it. In practice, as long as the tensions have the same value on opposite electrodes we only need to control two power supplies. Moreover, as the feature provided by the element is to focus the beam, we decided that absolute values of positive and negative tensions could be the same, just to simplify the control and without losing any advantage. This means that parameters A and B have the same value. By limiting the independent variables into one, we offer a very simple control interface. The only value the operator can modify is responsible for the focusing action of the element and is directly related with parameters A and B.

6.3. The control of a QP.

To control the QP a hardware system as shown in figure 6.4. has been implemented. Analog considerations are valid for the 58 quadrupoles, 25 steering quadrupoles and the five deflectors that compose the whole system and are driven in the same way.

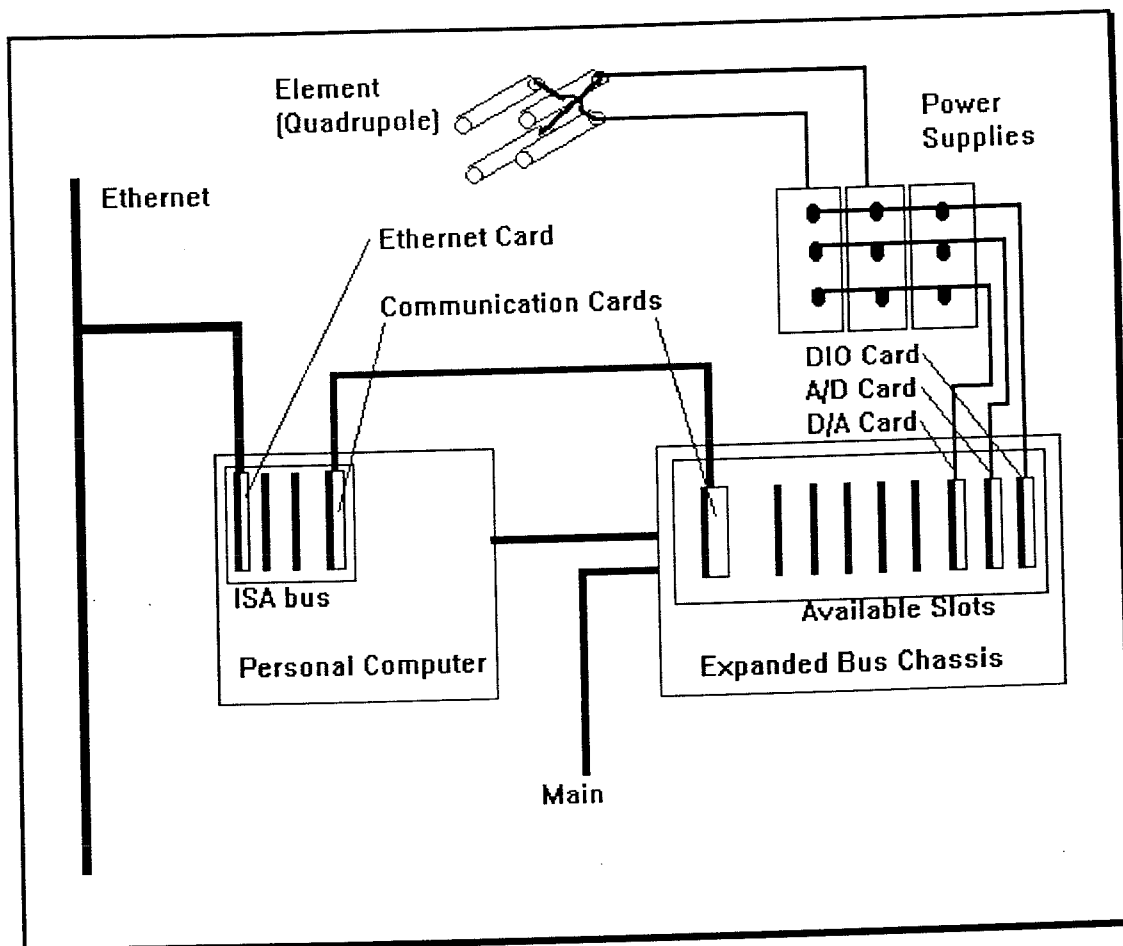


Figure 6.4. ISA expansion crates connected to the FEC.

The QPs system of ISOLDE can be divided in three different parts:

- The QP situated on the beam line;
- The power supply equipment used to power up the QP;
- The control system that enables the user to control the QP.

QP are used for symmetric focalization, and they are grouped in sets of one, two or three. We saw that the two positive as well as the two negative electrodes of each QP are on the same potential. Therefore only one positive and one negative power supply are necessary, the negative one being the slave of the positive one, in the sense that the negative power supply is directly driven from the positive one so that the number of channels necessary to drive the system is reduced to the half.

The old power supplies from the earlier project had to be reused. Those power supplies can give, as output, tensions varying from 0 to 3000 V. Positive and negative tension models are available and, for both, the choice of the value of the tension is done by varying a monitor tension between 0 and 10 V. There is also a retroaction monitor tension, representative of the tensions on the electrodes, which can vary between 0 and 10 V and which closes the loop. Last, a digital signal of 5 V specifies whether the power supply is on or not.

To control the power supply appropriately we then need a digital to analog converter to impose a value to the power supply, an analog to digital converter to read back the value of the power supply, and a digital input card to read if the power supply is working or not.

We decided to use for this purpose some commercial boards which can be fitted directly into the ISA bus of the FECs. This decision was influenced by the price and the availability of this kind of boards. Moreover, to keep maintenance easy and practical, we decided to buy some extension crates with the only purpose of not having the boards fitted directly in the computer but in an easy to open or change box.

The boards that have been chosen were all from Blue Chip Technology and they are the following:

- AOP8 : This card provides 8 analogue outputs in the range 0 - 10 V, at up to 10 mA per channel.
- AIP24 : This card provides 24 differential 12 bit analogue inputs. It is suitable for measuring voltages in the ranges 0 - 10, -5 to +5 or -10 to +10 V .
- DIO96 : This card provides 96 digital inputs. All channels could also be used for output but we didn't used this possibility.

In order to conceive the control system for this kind of elements it has been first decided to define all characteristics that were necessary to fully control a QP. Those characteristics have been called properties, and their values, completely defines the status of a QP. Some properties define the value of the tension on the electrodes, while some others defines the channel number and the addresses of the card to which the

element is connected. The software structure of each FEC, as we saw in section 4, consists in a local database loaded in working memory, which contains the values of the properties of all the elements connected to that FEC. Linked with this data table, we find a network listener, and functions expressly written to control the particular type of element connected to the FEC which are called Equipment Modules. Moreover on each FEC we find a local Nodal interpreter that allows access to the equipment from the FEC. This is used mainly by the equipment engineers to test modules locally. Each time the value of a property is changed, an access on the hardware results, and the same happens if the value of a property is requested. The access to the hardware is done by other modules, expressly written for the boards we mentioned just above, and which we are going to see later in this section.

The operator that want to control an element only see the relative knob and some parameters. When he changes one parameter he modifies the value of a property, and the console retrieve the address of the FEC controlling the element. Once this operation is done a Remote Procedure Call (RPC) is sent on the network and arrives to all the **RPClisteners** on the FECs. The requested FEC then catch the request and passes it to the correct module. The module verifies that the proposed value is within the correct range, retrieves the address of the card the QP is connected to and the channel number of the connection, weights the scaling factors to convert the value from Volts to Bits, and modify the tension, via some functions written for the specific boards. To make the system faster, during the reboot of the system all properties are loaded in working memory (RAM). This implies that all computers need to be rebooted each time a database is modified, but the system gain a lot in speed because it require less accesses to the server (the bottleneck of those systems is always the network).

6.4. Description of the properties.

The different properties implemented for a QP are:

- MINV as MINimum Value;
- MAXV as MAXimum Value;
- SCLR as SCaling factor Linear Read;
- SCCR as SCaling factor Constant Read;
- SCLW as SCaling factor Linear Write;
- SCCW as SCaling factor Constant Write;
- CCV as Control Current in Volts;
- ADRW as AddRess Write;
- CHNW as CHanNel Write;
- AQN as AcQuisitionN value;
- ADRR as AddRess Read;
- CHNR as CHanNel Read;
- STAQ as SStatus AcQuisition;
- ADRD as AddRess Read;
- CHNIO as CHanNel Input/Output;
- and CTRW as ConTRol Word.

6.4.1. The MINV and MAXV properties.

The purpose of the **MINV** and **MAXV** properties is to specify the minimum and maximum working values of the power supplies. There are several types of power supplies with working ranges varying from 0-1250 to 0-3500 V. By setting **MINV** to 0 and **MAXV** to 1250 the system will check that control values sent to the power supplies will remain in the range 0-1250 V.

6.4.2. The SCLR, SCCR, SCLW and SCCW properties.

SCLR, **SCCR**, **SCLW** and **SCCW** are the scaling factors. Their purpose is to fix a relation between the 12 bit digital value sent to the DAC card, or received from the ADC card, and the effective value we find on the QP. The 12 bit digital value ranges from 0 to 4096. The DAC card converts it in a low voltage tension which ranges

between 0 and 10 V. This tension is the monitor tension for the power supply which will output 0V for a monitor tension of 0V, and its maximum value (MAXV) for the 10 V monitor tension. In conclusion scaling factors are important to know which is the real tension on the QP while sending for each type of power supply the same range of digital values.

The formula that converts the desired tension on the element in the digital value to send to the DAC is:

$$\text{DigitalValue} = \text{SCCW} + (\text{SCLW} * \text{ValueInVolts}).$$

These properties are also very useful for calibration, because they permit to implement a software calibration procedure to calibrate the DAC and ADC cards.

6.4.3. The CCV, ADRW and CHNW properties.

CCV is the value in volts we send to the QP. By the use of the SCLR, SCCR, SCLW and SCCW properties this value is converted in the 12 bit digital value and sent to the correct channel of the correct DAC card. Each QP as we can see in figure 6.5. owns a DAC channel, an ADC channel and a DIO channel.

	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
	Name	FecName	MINV	MAXV	SCLR	SCCR	SCLW	SCCW	ADRW	ADRW	CHNR	CHNW	CTRW	ADRD	CHNO	CCV	AON	STAO
1	GPS OP040	POWER1	0	3500	1.17	0	1.17	0:1C0h	160h	2	2	2	100h	52	0x	x		
2	GPS OP050	POWER1	0	2000	2.0475	0	2.0475	0:1C0h	160h	3	3	3	100h	52	0x	x		
3	GPS OP170	POWER1	0	2000	2.0475	0	2.0475	0:1C0h	160h	4	4	4	1079bh	100h	54	0x	x	
4	GPS OP180	POWER1	0	1250	3.2768	0	3.2768	0:1C0h	160h	5	5	5	100h	54	0x	x		
5	GPS OP520	POWER1	0	3500	1.17	0	1.17	0:1C0h	160h	6	6	6	100h	54	0x	x		
6	GPS OP530	POWER1	0	3500	1.17	0	1.17	0:1C0h	160h	7	7	7	10b8bh	100h	56	0x	x	

Figure 6.5. Database of some of the QPs.

The property CHNW is used to know the channel of the ADC card, that is connected to the power supply. The property ADRW is used to define the base address of the DACs cards, as every card on the same bus has its own base address. It was previewed to use two FECs for the QPs (including also steering quadrupoles and deflectors). Infact all ADC DAC and DIO boards are located in four extension crates,

controlled by two FECs, called POWER1 and POWER2. FEC POWER1 is equipped with two extension crates and POWER2 with two others. Each extension crate is equipped with two ADCs cards for a total of 48 channels, five DACs for a total of 40 channels, and two DIOs for a total of 192 channels. Therefore every QP belongs to one FEC and the database specify which (Column 2 of Fig. 6.5.). As an example in table 6.1 we can see the addresses of the cards which equip the POWER1 FEC, divided into two crates.

POWER 1				
CRATE 1.1		CRATE 1.2		
<i>I/O Address</i>	<i>Component</i>	<i>I/O Address</i>	<i>Component</i>	
100	DIO-96	200	DIO-96	
120	DIO-96	220	DIO-96	
140	x	240	x	
160	AOP-8	260	AOP-8	
180	AOP-8	280	AOP-8	
190	AOP-8	290	AOP-8	
1A0	AOP-8	2A0	AOP-8	
1B0	AOP-8	2B0	AOP-8	
1C0	AIP-24	2C0	AIP-24	
1C4	AIP-24	2C4	AIP-24	

Table 6.1. The POWER 1 cards and their base addresses.

In conclusion, to reach a DAC card we need to know in which FEC the card is mounted, the base address of the card, and the channel of the card connected to the power supplies. All those information are stored in the database but when a FEC is initialized, it loads, for each element it has to control, all those parameters using them as properties. For this reason the information about which FEC is responsible of a QP is not a property.

6.4.4. AQN, ADRR and CHNR properties.

The **AQN** property is used to read back the monitor tension of the power supplies. As before the **ADRR** property is used to know the address of the ADC card, and the **CHNR** property to know the channel to which the power supply is connected.

6.4.5. The STAQ, ADRD, CHNIO and CTRW properties.

The **STAQ** property is used to know the status of the power supply. The power supply has an output channel that gives 5 V if it is working and 0V if not. This output channel is connected to a DIO card, and **CHNIO** specify to which channel of the DIO the power supply is connected, the **ADRD** property specify the base address of the card and **CTRW** is a control word used to initialize the DIO card to set all channels as input.

6.5. The QP equipment module.

The purpose of the QP equipment module is to control the 58 QP elements. Those elements can be controlled changing the values of their electrodes and reading back the values of the electrodes and the status of the related power supplies. When using consoles for operating the QPs, dialogue boxes have been made for controlling them. The dialogue box for a QP contains a scroll bar to change the value of the electrodes with the mouse, a windows where the desired tension can be introduced by typing it on the keyboard, and two other displays that show the read-back value of the tension and the status of the supplies. When the scroll bar, as an example, is moved by an operator, the dialogue box calls the function **D_POWP** in the FEC with some parameters defining what action to take. In our example, the action is to change the value on the electrodes.

When operating the QP from the Nodal for DOS program that we saw runs on the FEC, one has in addition to the same opportunities that the dialogue boxes in Windows

offer, the ability to change addresses of the cards, channel numbers, control words and so on.

The QP equipment module consists of a function called **D_POWP** which returns an integer value to the main program, some functions used by **D_POWP** that physically control the hardware, an array of structures that contains the database **POWP.CSV**, and some global variables.

The QP equipment module listing can be found in appendix C.

6.5.1. The function D_POWP.

The function head of **D_POWP** looks like this:

NodalError D_POWP (real far*value, int rwflag, int eqno, char *property)

In the file called **nodal.h** **NodalError** is defined to be integer, and **real** is defined to be double. Thus the function **D_POWP** is supposed to return an integer value to the main program. If the returned value is 0 the call of the **D_POWP** function didn't cause any errors, else the value corresponds to an error code that is sent to the console. The list of errors can be found in the file **errors.h** in appendix C.

The function **D_POWP** can be accessed from the **Nodal** for DOS program, and the QP system can be controlled via the commands **set** and **type** of the Nodal language.

Imagine we want to control a QP element named **GPS.QP170**. A look in Appendix D which contains the database **POWP.CSV** reveals that the equipment number of **GPS.QP170** is 4, and that the working range of the power supplies is between 0 and 2000 V. Set a tension of 1000 V on the **GPS.QP170** can be done, locally on the FEC, with the following Nodal command string:

set POWP(4, "CCV") = 1000

This Nodal command, once interpreted by the local Nodal interpreter, calls the function **D_POWP**:

NodalError D_POWP (real far*value, int rwflag, int eqno, char *property)

which gives the call:

D_POWP (1000, -1, 4, "CCV").

The nodal command is interpreted in the following way:

set : the **WRITE** flag is put in the **rwflag** of **D_POWP**;

POWP : calls the functions **D_POWP**;

4 : refers to the equipment number 4, of the element **GPS.QP170**;

"CVV" : defines the property in **D_POWP** to be used;

1000 : is the value we want **CCV** to assume.

If we want to read the status of the power supply of the element **GPS.QP170**, the Nodal string command is:

```
type POWP( 4, "STAQ")
```

where **type** implies that the **READ** code is put in the **RWFlag** of **D_POWP** which is called as follow:

```
D_POWP ( &value, 1, 4, "STAQ").
```

The output from Nodal when reading a QP status can be 1 if the power supply is **OFF** or 2 if it is **ON**.

When the **D_POWP** is called some local variables are created, and they are the integers **i** and **eq**. The integer **i** is just an ordinary counter. The integer **eq** is set to be **eqno-1** because the array of structures called **POWP_Table** goes from 0 to a variable called **MAX_POWP**. Before the **eq** is set to **eqno-1** a test is performed to test whether or not the variable **eqno** is within the range of legal number of QPs. If **eqno** is out of range the defined **illegal_equipment number** variable (in **errors.h**) will be returned to the user.

Below follows the explanation of the property **CCV** mentioned above. We will explain that one thoroughly, as all the others are built on the same basis.

```

/* Property CCV Current Control Value */
/* Read value from Data Table */
/* Write value to data table, scale using SCLW, write to hardware */
if (!strcmp(property,"CCV"))
{
    if (rwflag == WRITE) /* write value to hardware */
    {
        if (POWP_Table[eq].ADRW == -1)
        {
            return device_not_connected;
        }
        if (*value < POWP_Table[eq].MINV || *value > POWP_Table[eq].MAXV)
        {
            return out_of_range;
        }
        cc = OutValue(POWP_Table[eq].ADRW, POWP_Table[eq].CHNW,
                     POWP_Table[eq].SCLW, POWP_Table[eq].SCCW, *value);
        if (cc != 0) return cc;
        POWP_Table[eq].CCV = *value; /*UPDATE pow_tab */
        return 0;
    }
    if (rwflag == READ)
    {
        if (POWP_Table[eq].ADRW == -1)
        {
            return device_not_connected;
        }

        *value = POWP_Table[eq].CCV;
        return 0;
    }
    return illegal_read_write;
}

```

Figure 6.6. Listing of the CCV property of the D_POWP function.

The purpose of the CCV property (Fig. 6.6.) is to change the tension on the QP's electrodes. If the user calls the property CCV then the line:

if (!strcmp(property,"CCV"))

will be **TRUE** and the subroutine will be executed. The function **strcmp** returns zero if the strings are equal, a value that is less than zero if the first argument is smaller and a value greater than zero if the first argument is greater than the last argument.

The **rwflag** tells the equipment module whether the equipment should be written or read from. Writing means doing something physical on the QP, such as varying the tension of the power supplies and therefore the tension on the electrodes. Reading means just reading the last value which was sent and which is now stored in the **POWP_Table** in the FEC.

The handling of the **rwflag** is done with the lines:

if (rwflag == WRITE)

and

if (rwflag == READ)

If (**rwflag == WRITE**) is **TRUE** then the program checks if the card we want to write on exists, and if it is the case checks also that the value we want to write is within the permitted range defined by **MINV** and **MAXV**. If one of this tests fail the correspondent error is returned. Otherwise the program calls the function **OutValue**, which is a function that controls the hardware with the line:

```
cc = OutValue(POWP_Table[eq].ADRW, POWP_Table[eq].CHNW,  
              POWP_Table[eq].SCLW, POWP_Table[eq].SCCW, *value)
```

Parameters such as the base address of the card, the channel number, the scaling factors and the value to write are passed with this call.

If the function returns a zero then the value written on the hardware is updated in the **POWP_Table** and a zero is returned.

In the case the (**rwflag == READ**) is **TRUE** the value in the database, if existent, is simply given back and a zero is returned.

In the case neither (**rwflag == WRITE**) nor (**rwflag == READ**) are true, then an **illegal_read_write** is returned to say that an inexistent action was asked.

6.5.2. The special functions.

In Appendix C, following the function **D_POWP** some special functions taking care of the requests from the **D_POWP** function are listed. These functions provide writing the DAC cards (**OutValue** function), reading the ADC cards (**InValue** function) and reading the DIO cards (**InBit** function).

The **CCV** property we saw above called the **OutValue** function that we are going to describe below (Fig. 6.7.)

```

NodalError OutValue(int BaseAddress, int Channel, real SCL, real SCC, real value)
{
int pw,Low,High,Port;
NodalError cc;

    if (cc=nfix(value * SCL + SCC, &pw)) return cc;
    if ( pw >4095 ) pw=4095;
    if ( pw < 0 ) pw=0;

    High=pw>>8;
    Low=pw & 0xff;
    Port = BaseAddress + Channel * 2;
    outp( Port , Low); /* CHANNEL 0 LOW BIT*/
    outp( Port + 1 , High); /* CHANNEL 0 HIGH BIT */
    inp ( BaseAddress + 15); /* UPDATE STROBE OUTPUT */
    return 0;
}

```

Figure 6.7. The listing of the function OutValue.

This function is quite simple. Once the value **pw** is calculated, the functions checks that the value is within 12 bits, as the DAC board need a 12 bit information. The 12 bit information need then to be split in an 8 bit word plus a 4 bit word. This is done via a shift command for the 4 higher bits word, and via a mask for the lower 8 bits.

The address on which the correct channel is linked is calculated adding to the base address of the DAC card the double of the channel number. This value is stored in the variable **port**. The lower 8 bits are then transferred at the address stored in **port**, while the higher 4 in the following address. By reading the content of the address **BaseAddress+15** the value in the two address is converted in an analogue value and sent to the power supply.

7. Obtained results and critical evaluation.

7.1. How the system works.

In the previous sections we analyzed the system following an horizontal structure, that means, that each part of the system has been considered separately. If we want to see how the system works, it is better to identify and follow a vertical structure. This vertical structure is the information routing, that begins from the console to join the network, arrives to the FEC and from the FEC goes to the equipment. This is the routing that regards the command that from the console reaches the equipment. There is then all the way back, as for each command property, exists a read-back property useful to check if things are going as wanted. As long as we saw the QP we will now follow the information routing necessary to modify, as an example, the high tension on a QP and to read back the modified value. In Figure 7.1. we can see all the hardware involved in this operation and relative connections.

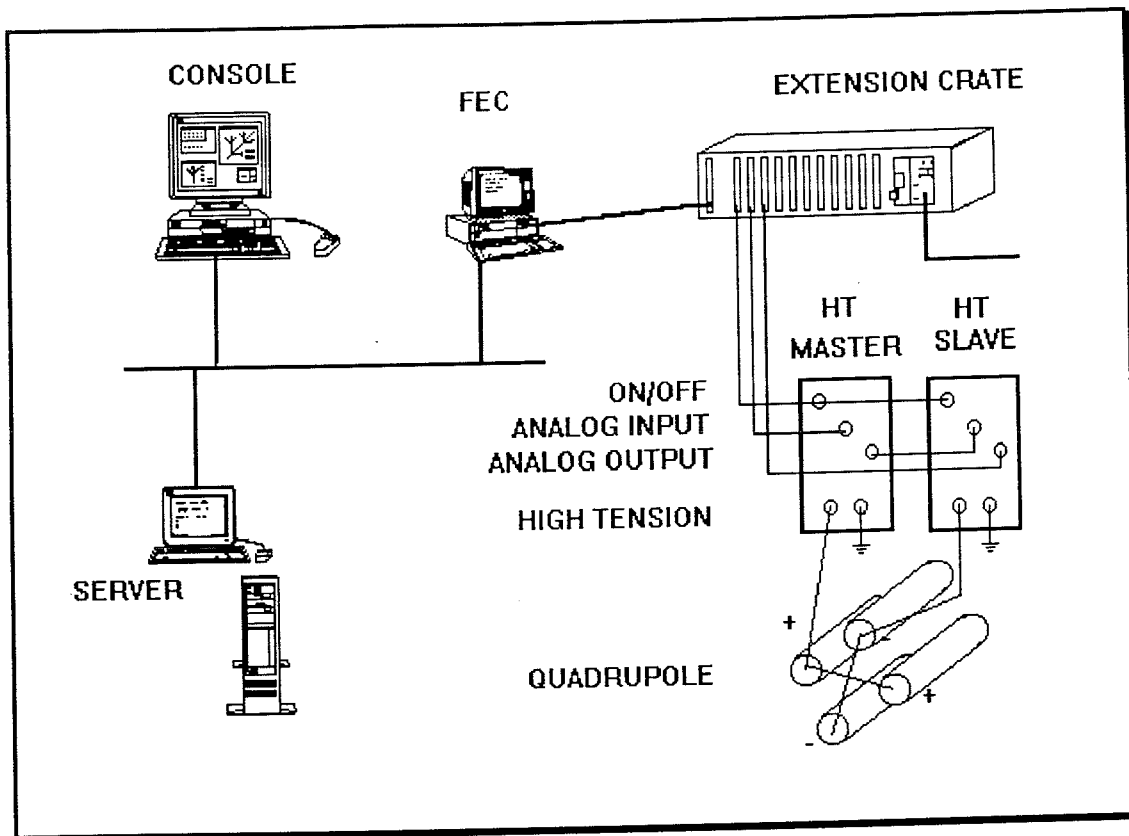


FIGURE 7.1. Hardware necessary to drive a quadrupole and relative connections.

7.2. Example.

7.2.1. Initialization of the FEC.

All devices that the **POWP** equipment module handles have addresses and initiation values that are stored in databases. The databases for all equipment modules are stored in the ISOLDE File Server. Upon startup of a FEC the database is read from the File Server and logged into the memory of the FEC by a Nodal for DOS function called **EMINIT**. In our case the **EMINIT** transfers the data from **POWP.CSV** to an array called **POWP_Table** of type **POW_DESC** which is created in the beginning of the equipment module (Refer to **POWP.C** in appendix C). When compiling the equipment

module and the rest of the files needed to create the executable file **FECNOD.EXE** that will control the QP system under Nodal, we can tell the compiler to make an executable that takes the argument **SYSGO**. In the DOS shell, typing **FECNOD SYSGO**, the program **FECNOD** will tell the Nodal interpreter to load and run the Nodal file **SYSGO.NOD** which looks like this:

1.10 type " Initializing the POWP equipment module"

1.20 EMINIT ("h:\eqp\powp\powp.csv)

1.30 type "Initiation done."

This small piece of Nodal code will be executed before the network listener is started. It will load all the data for operating the QP system. A quick glance in the **POWP.CSV** in appendix D reveals that the database doesn't just contain the name of the properties, addresses and initiation values, but also some comments like the first line plus the words **EqpNumber** and **FecName**. These names will not be sent as property-names to the equipment module because they are made of mixed cases, in comparison to the property names. Properties that should not be initialized can be empty or contain a "x" character.

7.2.2. Initialization of the console.

The initialization of the console is still database driven and consists in the initialization of the **RPCSERVER.EXE** program and of the application with which the element has to be controlled.

The **RPCSERVER.EXE** program loads, when started, three database files. They are called **EQPMOD.CSV**, **FECADDR.CSV** and **EQPNAMES.CSV**. Those files contain, respectively, data regarding the meaning of the weights of the bits of the control words (as 1 is **OFF** and 2 is **ON** and so on with **OPEN**, **CLOSED**..), the ethernet addresses of all computers involved in the control system and the names of all elements (see appendix D).

Each application controls a certain number of elements. Synoptics are Visual BasicTM applications and the names of the elements with information regarding the

properties they have are contained in the load procedure that is called from the Visual Basic™ interpreter, when the application is started. In Figure 7.2. an example of the data of a load procedure is given.

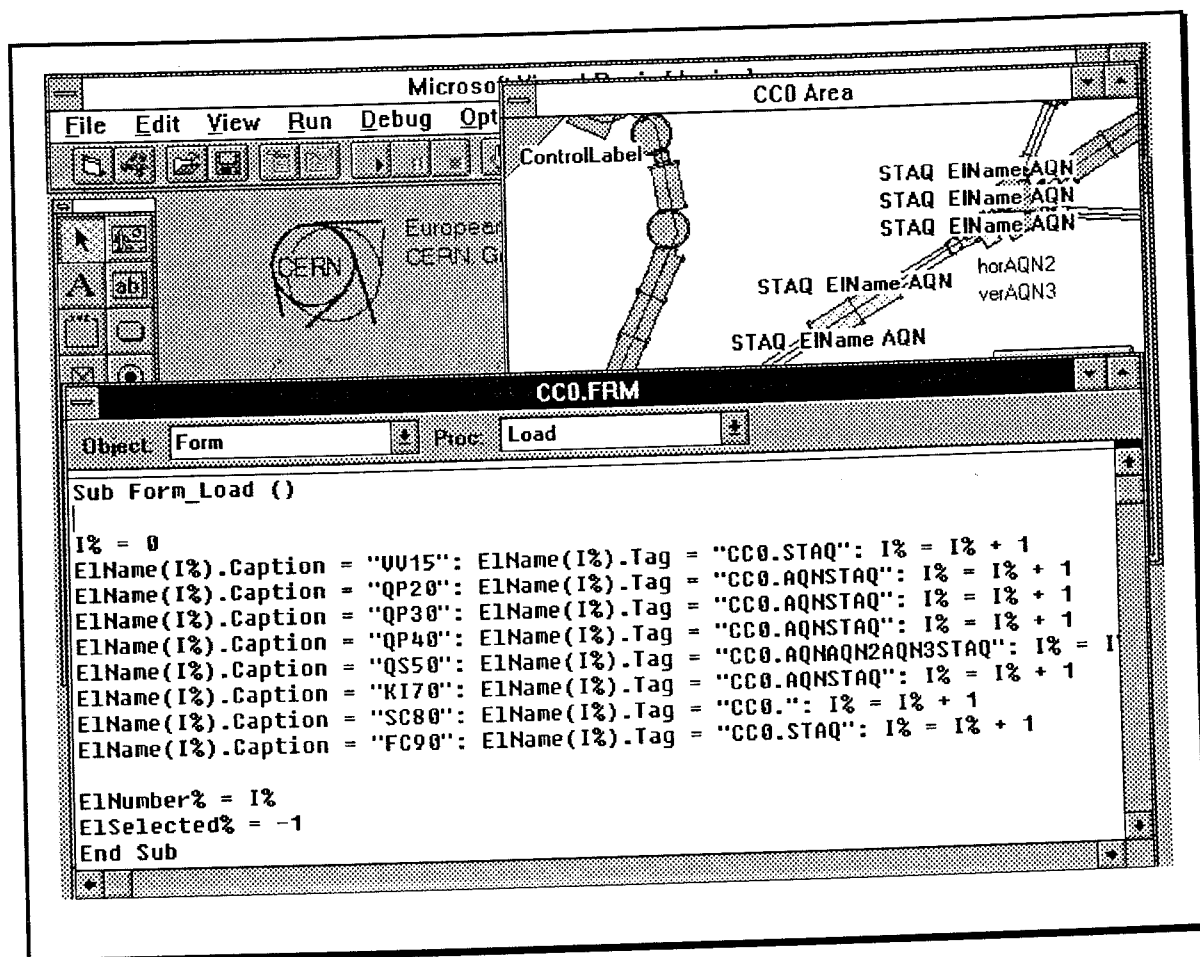


FIGURE 7.2 The load procedure of a Visual Basic™ application.

7.2.3. The information routing.

The user of the facility works in a transparent way on the control system. Suppose the operator wants to modify the tension of the quadrupole **GPS.QP170**. The suffix of the name of the element suggest that this element is part of the General Purpose Separator. To control that element the GPS synoptic has then to be opened. Once called the synoptic will show a view of the GPS section on the upper right of the

screen of the console. It is now necessary to zoom on the area that shows the required element. The general view proposes (as in Fig. 7.3) the names of the possible zooms.

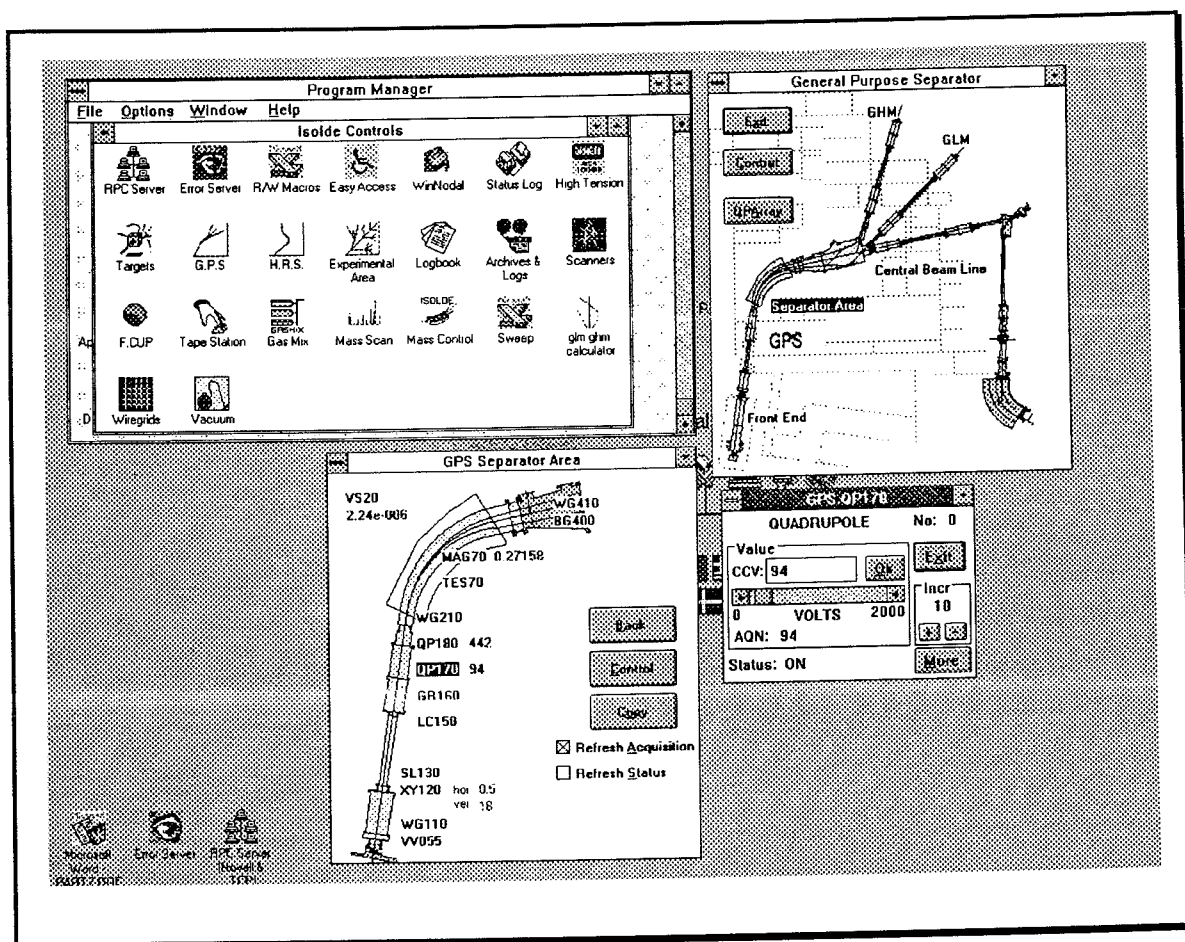


Figure 7.3. The GPS application used to control the GPS.QP170 QP.

The operator visually recognize the area it has to select and ask the control of that area. The new synoptic appears in the lower left region of the screen and contains the names of all elements. There is also the possibility to ask for the status and the tensions in the elements by clicking on the options **Status Refresh** and **Acquisition Refresh**. But, to have the control of an element, the operator should ask the control panel of the element by double-clicking with the mouse the element name or by selecting it and clicking the **Control** button. The **RPCSERVER** program creates then a control panel on the lower right part of the screen and this permits the operator to control the

element. Before the control panel is created the **RPCSERVER** program does a complex operation to see:

- On which FEC the element is, by consulting the **EQPNAMES.CSV** database it has loaded in RAM;
- Which is the ethernet address of the FEC, by consulting the **FECADDR.CSV** database in RAM;
- Which type of control panel the element requires, again by reading the **EQPNAMES.CSV** file;
- And finally to ask the working range of the element, the status and the actual tension of it, by sending requests to the interested FEC.

In Fig. 7.3. we can see how a control panel looks like. The operator can read the status of the element (**ON\OFF** for a QP), and can read and write the tension value.

Suppose the actual value of the high tension is 500V and that the operator wants to modify it to 1500V. The ways he has to do so are to write directly the requested value in the window or to push with the mouse the arrow of the tension cursor. Once a new value is established the **RPCSERVER** program send on the network a Remote Procedure Call (RPC), which is a coded message that ask to a defined FEC (by the **FECADDR.CSV** file) to modify a defined property value (by the load procedure of the application) of a defined element (checked in the **EQPNAMES.CSV** file).

The RPC is listened by all the other computers, but interpreted by the correct FEC that will call the **POWP** procedure we analyzed in section 6, passing all the necessary parameters such as the element number to control, the property to modify, the code to specify we want to write a value and not to read, and the new tension value.

The **RPCSERVER** refresh with an interval defined by the user, all the values of the active control panels.

7.3. Applications.

In order to give an idea of the possibilities offered by this control system, in this chapter a brief description of each application will be given. The applications that we are going to see are: **Targets, GPS, HRS, Experimental Area, QPArray, Vacuum, GasMix and FCup.**

7.3.1. The Targets application.

The **Targets** application (Fig. 7.4.) is used to control the different targets that can be mounted as ion source. The first window that appears when we call the Targets application shows the possible targets. The actually mounted target has its name written with green characters, while the others have red text. The control window of the targets can be called by double-clicking the picture or the name of the target or by selecting the name and asking the control with the **Control** button.

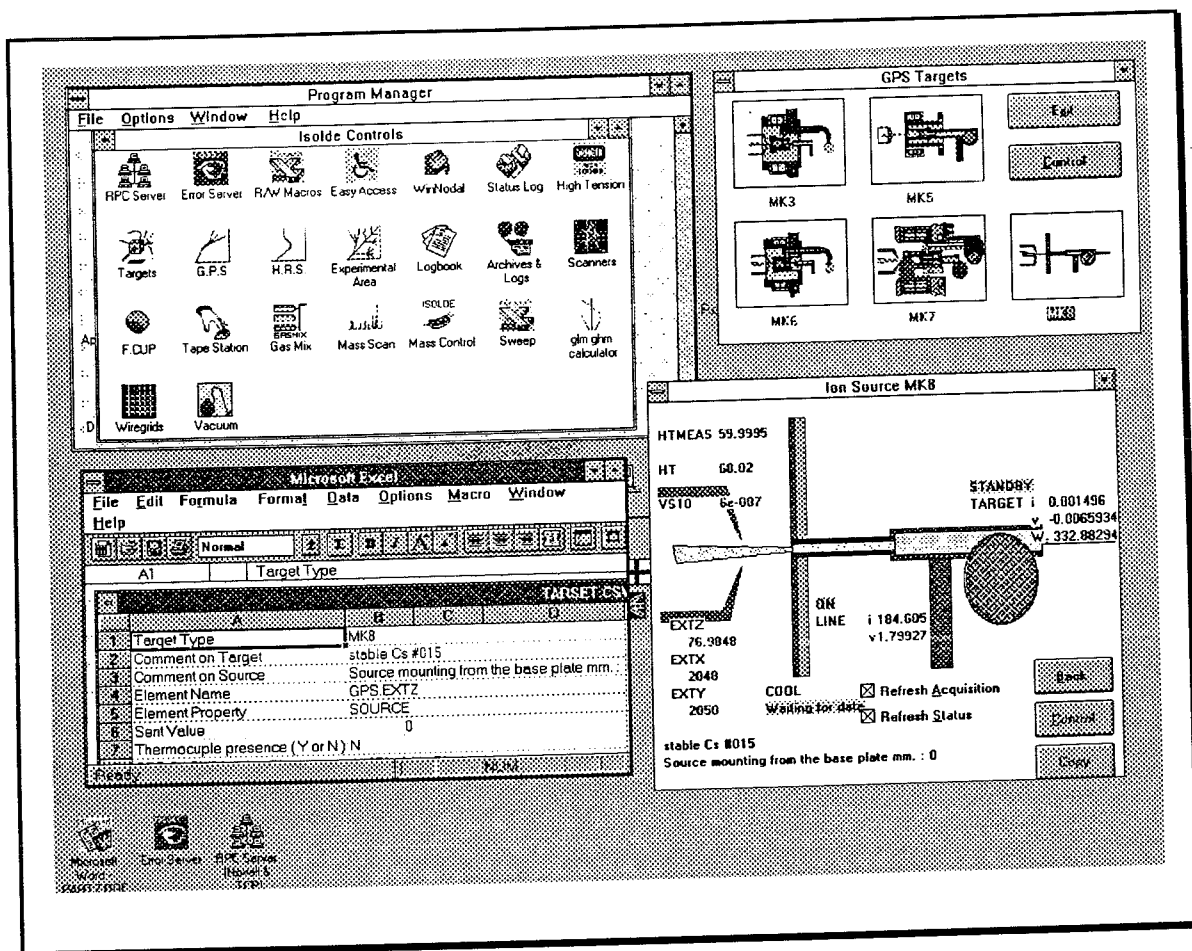


Figure 7.4. The Targets application and the database file TARSET.CSV.

Once the new window is loaded it is possible to select the element and ask for its control panel, or to ask the status of all the elements to be reported in the picture. In the bottom of the picture a brief description of the target is given to let the user know type and characteristics of the source in use. This information note as the information saying which target is mounted is stored in the database file **TARSET.CSV** as shown in figure 7.4.

7.3.2. The GPS, HRS, and Experimental Area.

Those three applications are in fact the same application that was split in three parts to avoid screen competition. Each part is dedicated to an area of the system, as their

names suggest. We saw their functioning in the example and the only thing we have to add is that from this application it is possible to call the **QPAarray** application.

7.3.3. The QPAarray application.

The **QPAarray** application was asked by the users for data logging and retrieving. When the user reaches an optimal calibration of the beam for his purpose, he would like to save the system configuration, to be able to retrieve it in the future. The principle of this application is that the user should write in a database file all the elements name that are interested in his calibration (those name changes in function of where the experiment is mounted in the experimental area because different sections of the facility are used), and call then the application. The application is shown in figure 7.5.

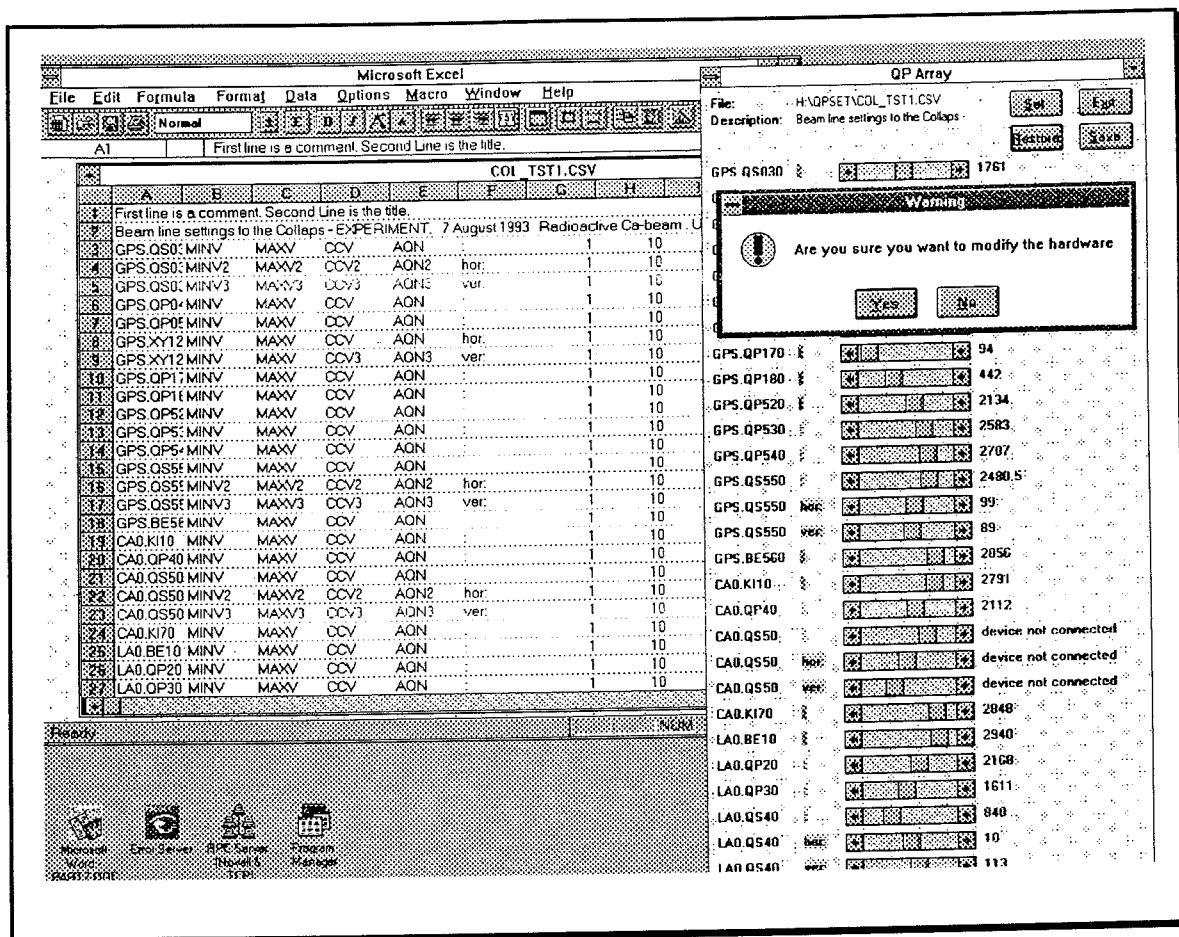


Figure 7.5. The QPArray application and a database file.

The application permits to read the description of the saved configuration, which is the first line of the database file, and once one file is selected it shows a window which contains a scrollbar and the principal properties of all the elements selected. The user can use the scrollbars to trim the beam and can then save the configuration pushing the **Save** button. By pushing the **Restore** button the saved configuration is restored. Confirmation message boxes avoid involuntary manipulation (Fig. 7.5.).

7.3.4. The Vacuum application.

The **Vacuum** application was written to control the vacuum system that maintains a very low pressure inside the sections. Valves are used at the edges of each section to permit the system to work while some experiments are mounted or dismantled. This

application (Fig. 7.6.) shows the names of all the sections and by double-clicking this names the user get the control panel that controls the pump of that section.

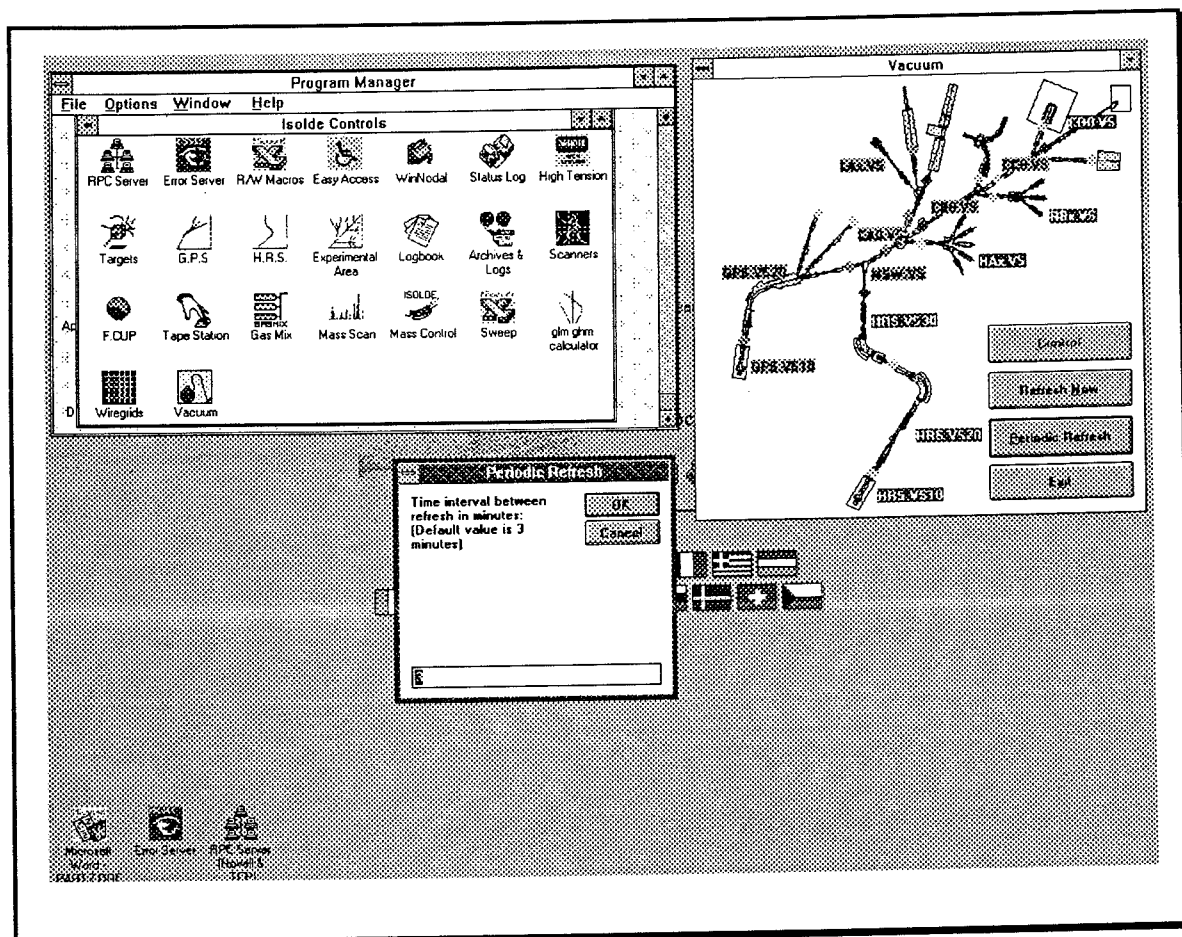


Figure 7.6. The Vacuum application.

On the layout there are the valves which are represented by small squares. The color of the square follows the convention that red means that the valve is **CLOSED**, green is **OPEN**, and yellow is **INVALID** (which means that the valve is blocked between the two possible position). The user can also select the refresh time of the picture as to have an immediate refresh.

7.3.5. The GasMix application.

The scope of the **GasMix** application is to permit the formation of a mix of gases in the targets in a remote controlled way (as the target area is strongly radioactive). A

database file contains the name of the types of gas that is available (Fig. 7.7.), and an options selection permits to control the valve to open. The database file is loaded with the program.

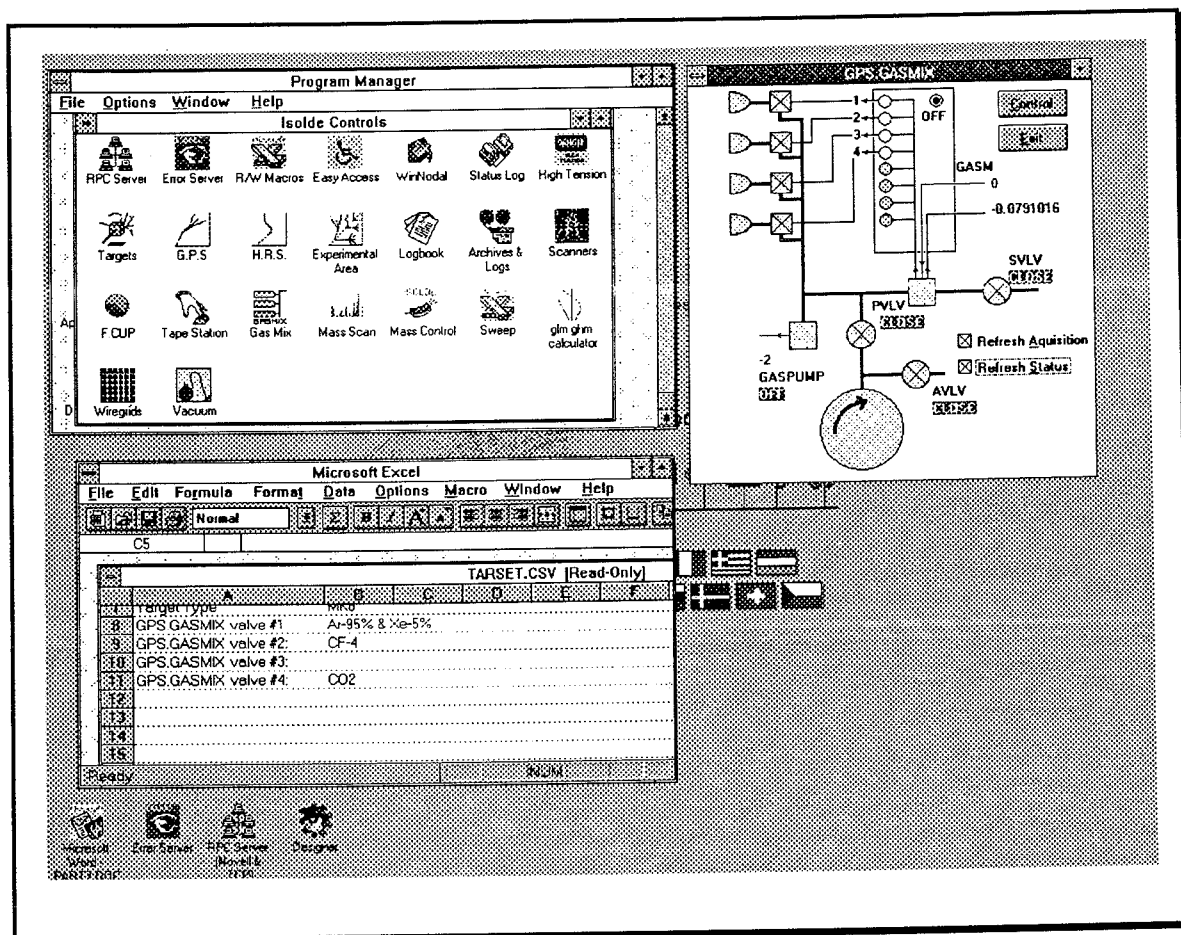


Figure 7.7. The GasMix application and the database file.

7.3.6. The Faraday Cup application

The **F.Cup** application was asked from the users to use in conjunction with the **QPArray** application to trim the beam. Once the application is called, the user can select, between all the installed Faraday Cups, which to connect to. The Faraday Cup can be **IN** or **OUT**. If the selected one is **OUT** then a text saying this appears. Otherwise appears a pie proportional to the beam intensity (Fig. 7.8.)

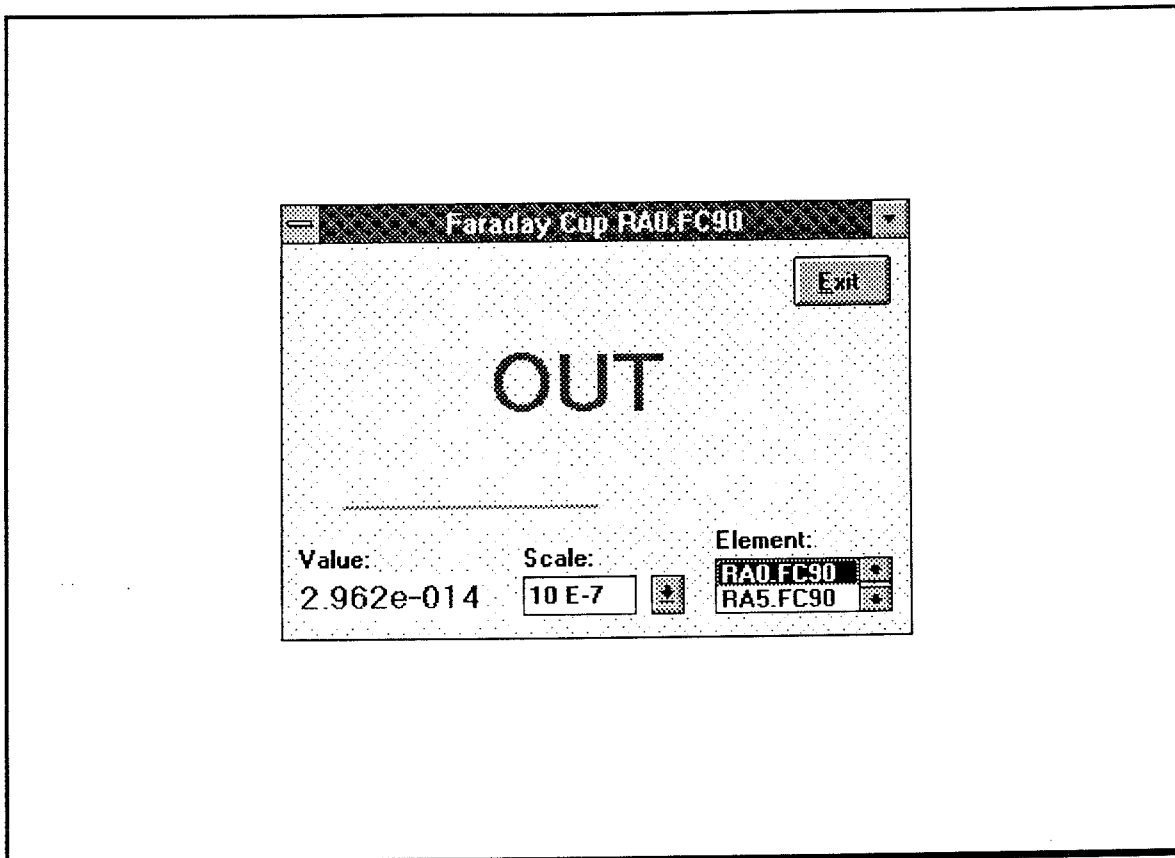


Figure 7.8. Example of the Faraday Cup application.

7.4. Critical evaluation.

The ISOLDE control system project implemented Olivetti PCs as workstations. They feature powerful enough CPU, high or very high resolution monitors and, very important, they run an operating system which allows the user to interact with a graphical environment, a Graphic User Interface (GUI). Last but not least, they are up to ten times less expensive than other workstations.

Another major element of the system is the integrated network hardware and software. These computers are designed to be part of a network of computers. For this reason it was simpler and cheaper to incorporate them into a distributed control system. Moreover, as those computers are inexpensive and easy to network, the

control system gains a lot in flexibility. For example, a simple way to increase system performance is to dedicate one or more PCs solely to the heavier applications. This solution has been adopted when we decided to install three consoles, just to avoid the usual bottleneck of more applications competing for display screens.

Using several, but economical computers, system downtime can be minimized, because spare machines and consoles are now economically practical and users of the system can monitor operations or examine data in their offices, using workstations that tie into the control system (At CERN 1600 PCs are connected to the network). All the advantages we identified are primarily function of cost. For this reason the choice of the less expensive family of workstations (the PC family) amplified these advantages. The result is that every ISOLDE user has a PC in his office, sometimes because he just had it before, other times because he could afford to buy one.

All this is possible because of the positive conjuncture the PC market offers today. A mix of market leader products permits today to build a cheap, powerful and robust control system. Last but not least the disponibility of developing software with respect to the human interface, permitted to put less programming effort than into traditional systems.

Of course the system presents also some drawbacks. The major problem we experienced is reliability. The harsh industrial electrical network of the experimental area caused some problems for the power supplies of both PCs and expansion crates, which broke down in some occasions. We also registered some hard disk failures, but we do not know if the cause is to be attributed to the age of the FEC computers which are old AT-286 computers. The same happened with some ISAbus cards, which are not optoisolated and are, as a result, sensible to tension spikes in the channel lines. The software proved to be robust, and only few bugs have been encountered, principally due to the friendliness of the developing system we used. One lack of the system is that the operating system used (MS-DOS + Windows) do not implement a preentitive multi-task. Thus, if an acquisition process is for some reasons blocked, then it is

necessary to wait the time-out period to have back the control of the console and this seems to be annoying for the users.

For the future we rely on the progress that the PC family will achieve, and on the progress of operating systems and development environment. In the short term we can think to the Intel PentiumTM, which will give a real speed boost to the PC family, for what regard the hardware. Considering operating systems Microsoft Windows NT, will be on the market in the short term, proposing integrated network features and the possibility to exchange DDE messages between different computers. This means that the **RPCServer** could be no more necessary, with improvements in operation speed and, maybe, communication modes. As an example it is envisaged to have the possibility to communicate between two FECs to synchronize them. Today this is not possible, but it will certainly be in the future. Developing systems are evolving very fast, and again in the short term we can rely on two new products. One is the C version of Visual BasicTM, that will offer a compiled code with sensible speed improvements. The second is Microsoft AccessTM, a powerful database with the possibility to add interpreted code, windows and commands. This software permits to create control systems with integrated database capabilities.

8. Conclusions.

Traditional control systems frequently results in "home grown" products which remain incomplete and are overtaken by the rapid advances of the massive industrial base of software.

Conscious of this consideration, in implementing this control system, we tried, as far as possible, to avoid this situation. The obtained results of our work can well be synthesized in the following few points:

8.1. Simplicity.

Using the same type of computer (ISA PC compatible) at all level (console and FEC) of the control system leads to a strong uniformation. The advantage is reinforced by using the same operating system as the office machines.

All the persons involved in the design of this control system, understand it completely and are able to solve any problem at any level, either at the front end, either at the console. This global view on an uniform system reduces the skill necessary to cope with different parts of the system, because the approach is always the same, data driven from the Excel Database.

8.2. Development time.

The use of well tested, market leader products (hardware and software) has reduced the development to a strict minimum. The very few necessary development done with the available tools have shown that:

- Development is very quick because of the user friendliness of the tools.
- Informatic skills are not necessary: Physicists and equipment responsible can write their own programs after very little training.

In particular, the time to develop the Application programs, that is normally considered as being the largest investment in terms of manpower is very small. For

example, it does not take more than 10 minutes to build a simple application using Excel to display graphically several parameters acquired from the control system.

8.3. Integration.

The ISOLDE control system, although completely independent in case of problems, is transparently connected to the office network and all its services are available.

Beside providing an access to the ISOLDE control system by nearly 1600 points at CERN (the number of PCs installed), it is in the user culture by providing the same user interface to control applications as to any administrative or scientific applications. It is of course possible to Copy/Cut/Paste between any applications, for example, the Excel chart acquiring the status of the ISOLDE magnets can be pasted into Microsoft Word to produce a written report. Development of control application from any office is possible without the installation of additional hardware.

All A3/A4 laser printers (including the color ones), plotters, scanners, disks, software and data of the office network are available, including the support.

By relying on this infrastructure, the information can be easily distributed CERNwide as statistics and/or electronic logbook. Being the ISOLDE file system shareable between different platforms and operating systems (DOS, Macintosh, Windows, OS/2, UNIX), all data in Excel format can be retrieved from any user on the site.

8.4. Speed.

All benchmarks done with the Olivetti 386/25 computers has shown that this computers are fast enough to control large processes where dozens of parameters form different FECs are involved. The margin can be enlarged in this area by using Intel 80486 based 33 MHz machines that are much faster.

The speed of the control system is often limited by the network used. The Novell network based on ethernet give us a performance of more than 100 Kbytes/second

transfer rate on the busy CERN ethernet. A typical Remote Procedure Call with a search in the element database to find which FEC and which equipment module to call takes less than 30 milliseconds.

8.5. Cost.

The market of Personal Computer in the world is roughly of 60 Millions units and with a strong competition between manufacturers. The PC production is more than 10 Millions/unit per year with performances increasing by a factor of 2 every year.

It's a world wide standard, with binary compatibility, with major European involvement such as Olivetti, Philips, Bull and Siemens. A crate, made in Europe, to control 48 power supplies costs about 12 KCHF. The cost per control channel is 9 CHF per digital bit, 55 CHF per ADC channel and 155 CHF per DAC channel.

8.6. Resonance.

Few words have finally to be spent about the attention that this system received. The system was presented at the Third International Conference on Accelerators and Large Experimental Physics Control Systems that was held at Tsukuba in Japan in November 1991, and at the "Centro di Cultura Scientifica Alessandro Volta" in Como, Italy. As a result of these presentations this control system has been adopted from the Daresbury Laboratory at Warrington, Cheshire, and from the DESY experiment in Hamburg.

References.

- [1] The PS and SL Control Groups, R.Rausch, C. Serre, Editors, " PS/SL Controls Consolidation Project", Technical Report, PS/91-09, CERN, Geneva, Switzerland, April 1991.

- [2] B. Kuiper, "Accelerator Controls at CERN: some Converging Trends", Nuclear Instruments and Methods in Physics Research A293 (1990) 308-315.

- [3] K.H. Kissler, R. Raush, "New Control Architecture for the SPS Accelerator at CERN" International Conference on Large Experimental Physics Control System, Tsukuba, Japan, 11-15 November, 1991.

- [4] L. Jirdén, "Personal Computers in the LEP Control System", Nuclear Instruments and Methods in Physics Research A293 (1990) 221-224.

- [5] Commission of the European Communities, "Guide to Internationally Recognized Hardware Interfaces with Tutorial", Report 11079, European CAMAC Association, Université Catholique de Louvain, Belgium, 1988.

- [6] R. Raush, "Bus d'Interconnexion Multi-Maître, Multi-Chassis VME", Forum sur la Microinformatique en Physique Nucléaire et Physique des Particules, Collège de France, Paris, France, 19-20 Septembre, 1983.

- [7] P. Burton, "Industrial Field Bus, Standardization & Conformance Testing", CEN/CENELEC Conference on Industrial Local Area Networks, Palais des Congrès, Brussels, Belgium, 22 November, 1988.

- [8] P.S. Anderssen, "Personal Computer in Accelerator Control", Computer Physics Communications 50 (1988) 89-99.
- [9] "Personal Computer Bus Standard P996", Draft D2.00, Copyright IEEE Inc, January 18, 1990.
- [10] "ISA Bus Specification and Application Notes", Copyright Intel Corporation, January 30, 1990
- [11] "Technical Reference Guide, Extended Industry Standard Architecture Expansion Bus", Copyright COMPAQ COMPUTER Corporation, 1989.
- [12] C.F. Parkman, "VICbus: VMEbus Inter-Crate Bus, a versatile Cable Bus", Real-Time 91 Conference, Jülich, Germany, 24-28 June, 1991.
- [13] K.H. Kissler, R.Rausch, "New Control Architecture for the SPS Accelerator at CERN", Paper presented at the Third International Conference on Accelerators and Large Experimental Physics Control Systems (ICALEPCS'91), Tskuba, Japan 11-15 November 1991.
- [14] C. Eck, "Standardization of Real-Time Software POSIX 1003.4", Real-Time 91 Conference, Jülich, Germany, 24-28 June, 1991.
- [15] R. Rausch, Ch. Serre, "Common Control System for the CERN accelerators", Paper presented at the Third International Conference on Accelerators and Large Experimental Physics Control Systems (ICALEPCS'91), Tskuba, Japan 11-15 November 1991.

- [16] R. Rausch, "Controls of the Large European LEP and SPS Accelerators Based on the 1553 Field Bus", Conference on MIL-STD-1553-B and the Next Generation, London, United Kingdom, 29-30 November, 1989.
- [17] The LEP/SPS Control Group, "The LEP/SPS Access to Equipment", LEP Controls Note 54, Version 2.0, CERN, Geneva, Switzerland, 15 March, 1988.
- [18] W. Heinze, "Driving serial CAMAC systems from VME crates", Paper presented at the Third International Conference on Accelerators and Large Experimental Physics Control Systems (ICALEPCS'91), Tsukuba, Japan 11-15 November 1991.
- [19] C.R.C.B. Parker, M.E. Allen, H.C. Lue, C.G. Saltmarsh, "A New Data Communication Strategy for Accelerator Control Systems", Nuclear Instruments and Methods in Physics Research A293 (1990) 239-242.
- [20] R. Raush, "Standards in Computerized Control" Paper presented at the VIIIth International Symposium on Modular Information Computer Systems and Networks, Academy of Science of the USSR and ESONE Committee, Dubna, USSR, September 10-13, 1991.
- [21] E. Kugler et al., "The new CERN-ISOLDE on-line mass-separator facility at the PS-Booster", Nuclear Instruments and Methods in Physics Research B70 (1992) 41-49.
- [22] B.W. Allardyce et al., "ISOLDE at the PS-Booster", Paper presented at the XVth International Conference on High Energy Accelerators, Hamburg, Germany, 20-24 June 1992.

- [23] O.C. Jonsson et al., "The control system of the CERN-ISOLDE on-line mass separator facility", Nuclear Instruments and Methods in Physics Research B70 (1992) 541-545.
- [24] A. Ogle, J. Ulander, I. Wilkie, "Experience with Workstations for Accelerator Control at the CERN SPS" Papers presented at the International Conference on Accelerator and Large Experimental Physics Control Systems", held at Vancouver, British Columbia, Canada October 30- November 3, 1989.
- [25] M. Boutheon, F. Di Maio, A. Pace, "General Man-Machine Interface used in Accelerators Controls: Some Applications in CERN-PS Control Systems Rejuvenation", Paper presented at the Third International Conference on Accelerators and Large Experimental Physics Control Systems (ICALEPCS'91), Tskuba, Japan 11-15 November 1991.
- [26] E. Theil, V. Jacobson, V. Paxson, "The impact of new computer technology on accelerator control", Presented at the 1987 IEEE Particle Accelerator Conference, Washington, DC, March 16-19, 1987.
- [27] R. Feynmann, "The Feynmann Lectures on Physics" Vol. II, Part I, Masson editore, 1963.

Appendix A : Some information about CERN.

Appendix A : Some information about CERN.

Table of contents.

A.1. CERN..... II

A.2. Budget and Member States. II

A.3. Business..... III

A.4. Human Resources. IV

A.5. Industries Involved. VI

A.6. Accelerators..... VII

A.1. CERN

CERN is one of the world's largest scientific laboratories and an outstanding example of international collaboration. The acronym comes from an earlier French title - Conseil Européen pour la Recherche Nucléaire - while the actual name of the laboratory is European Laboratory for Particle Physics. With its 600 hectares of extension it straddles the French-Swiss border just west of the city of Geneva.

A.2. Budget and Member States.

CERN's annual budget (945.47 million Swiss francs in 1992) is mainly provided by its eighteen Member States from Europe. These are Austria, Belgium, Czech and Slovak Federal Republic, Poland, Finland, France, Germany, Greece, Hungary, Italy, Netherlands, Norway, Denmark, Portugal, Spain, Sweden, Switzerland and United Kingdom. Observers are Turkey, Yugoslavia, Israel, Russian Federation, Commission of the European Communities, UNESCO.

Below is given a table showing the normalized contributions of the Member States.

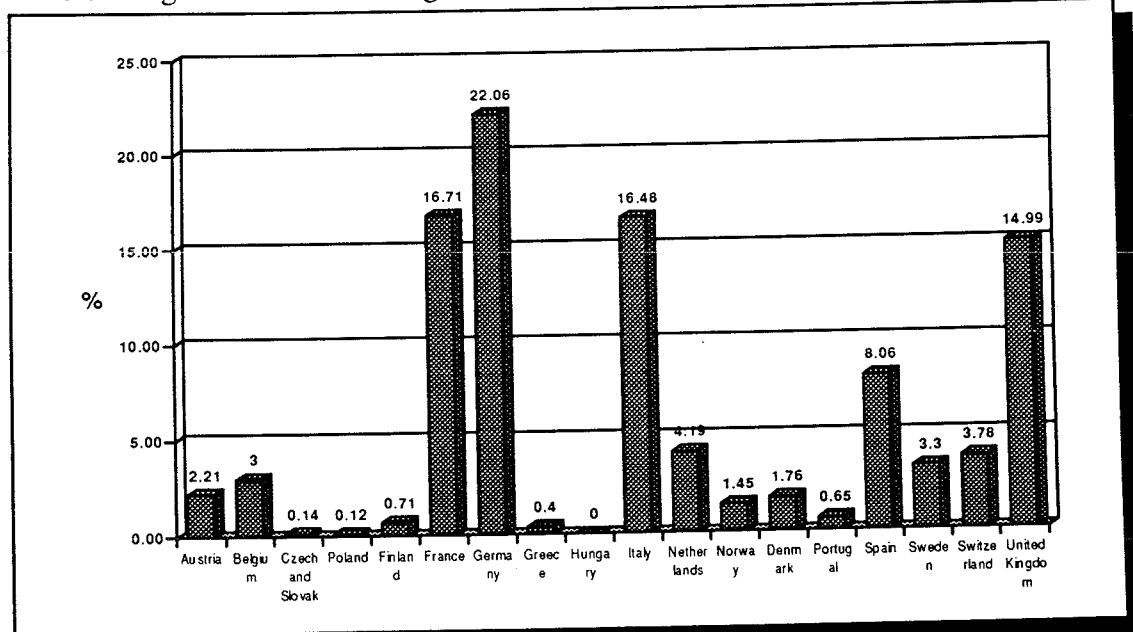


Figure A.1. Member States and their normalized contribution (%).

This budget covers expenditures of research, accelerators and experimental areas, technical support and administrative support. The graph below shows the contribution of the budget on these areas.

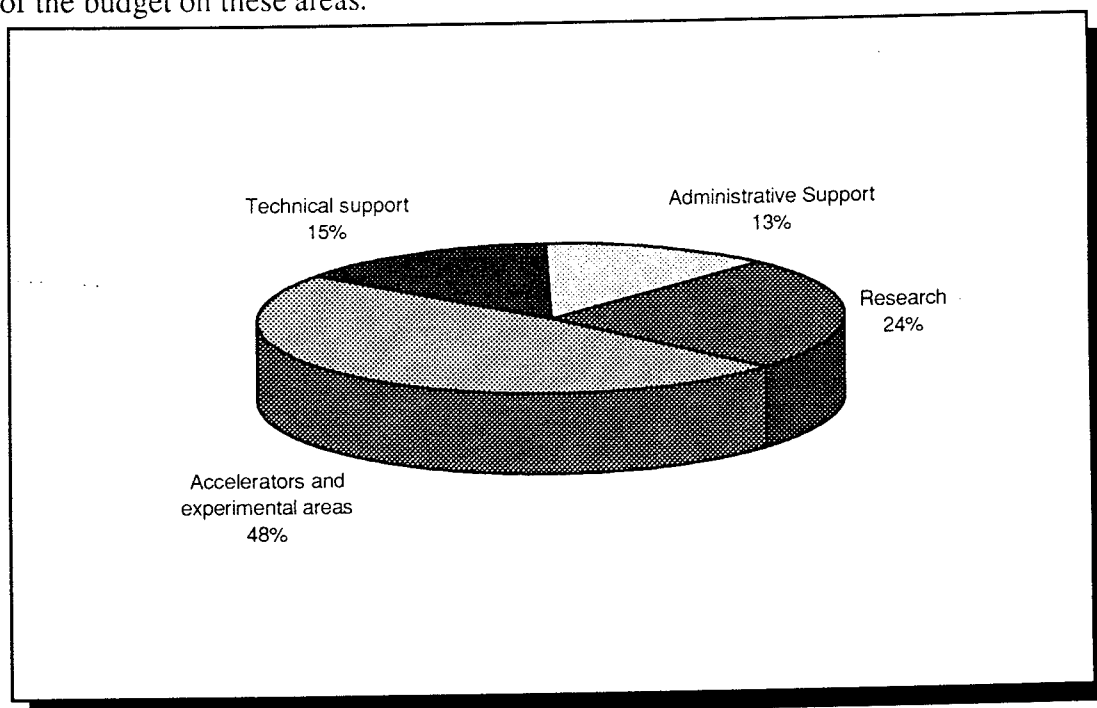


Figure A.2. Expenditure distribution.

A.3. Business.

Article II of the Convention stipulates that CERN shall provide for collaboration among European states in particle physics research of a pure scientific and fundamental character, that it shall have no concern with work for military requirements, and theoretical work shall be published or otherwise made generally available.

In practice CERN's tasks are:

- to ensure the construction and operation of particle accelerators and their necessary ancillary apparatus;

- to define a programme of activities and to submit it for approval to the Council which is composed of representatives of its Member States;
- to organize and sponsor international co-operation in this field of research, particularly with and between Member States laboratories.

CERN's business is then pure particle physics science. However, even in the short term this research, with its stringent demands for accuracy and ultra-fast response, frequently pushes modern technology to the limit, providing a catalyst for technological innovation with important spinoff in other fields. New techniques emerge such as scanners for medical diagnosis, radioactive tracers, ion implantation, radiocarbon dating, etc. In the long term, this new knowledge could lead to unimaginable technological advances. Electricity, X-rays, nuclear energy and lasers are just a few examples of laboratory discoveries from previous generations of this kind of research which quickly made their mark on technology and went on to become part of our everyday world.

A.4. Human Resources.

To design and build CERN's intricate machinery and to ensure its smooth operation, to help prepare, run, analyse and interpret the complex scientific experiments, and to carry out the myriad of tasks in such a large and special organization, CERN employs over 3000 people, encompassing a wide range of skills and trades: engineers, technicians, craftsmen, administrators, secretaries and workmen. Below is given a Figure that shows the type of job exploited by people employed at CERN.

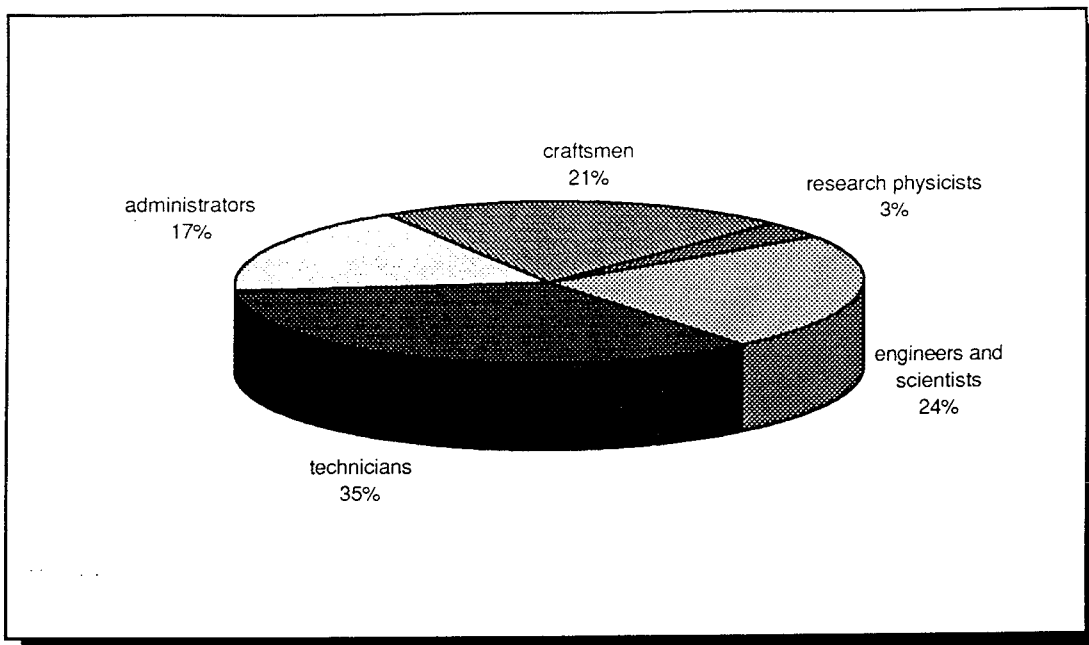


Figure A.4. Staff (Total 3126 as of 31st December 1991).

CERN's research programme, based on its unique range of big machines, attracts scientists from the Member States and from further afield - the United States of America, Latin America, Japan, Canada, the Russian Federation, the People's Republic of China, Israel and India. While working at CERN, most of these visiting scientists are paid by their university or research institute. In 1991, 4625 users have been coming from 272 institutes or universities and can be divided as follows.

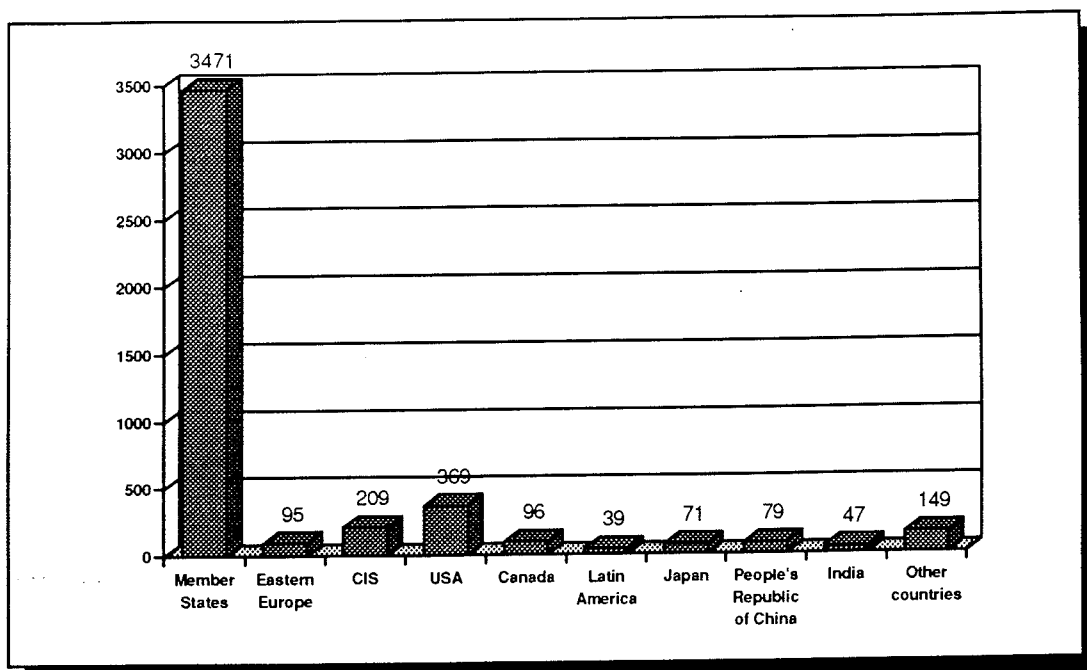


Figure A.5. Number and countries of experimental physicists (Users).

A.5. Industries Involved.

CERN's and the Member States' success in controlling project costs owes much to the Organization's purchasing rules. CERN has its own purchasing policy, governed by clear procedures set out in its Financial Rules. Their purpose is to ensure that all goods and services required are obtained at the lowest market price, with preference being given to European industry.

The simple graph of the distribution of purchases, however, is not representative on the competitive level of a country. Several types of goods need to be purchased locally, independently of its convenience. This is the case of electricity, services, maintenance and transports. For this reason France and Switzerland profit the most from CERN location. A relevant information can be desumed analyzing contracts of a big project. LEP with its final cost of more than 1200 millions of Swiss Francs can give a good idea of contracts distribution. In Figure 3.6 we can compare the differences between the total expenditures of one year (1991 with a total of ~ 500 MCHF) and the

expenditures for the LEP project (Relative to the 400 biggest industries involved covering the 80% of the total costs).

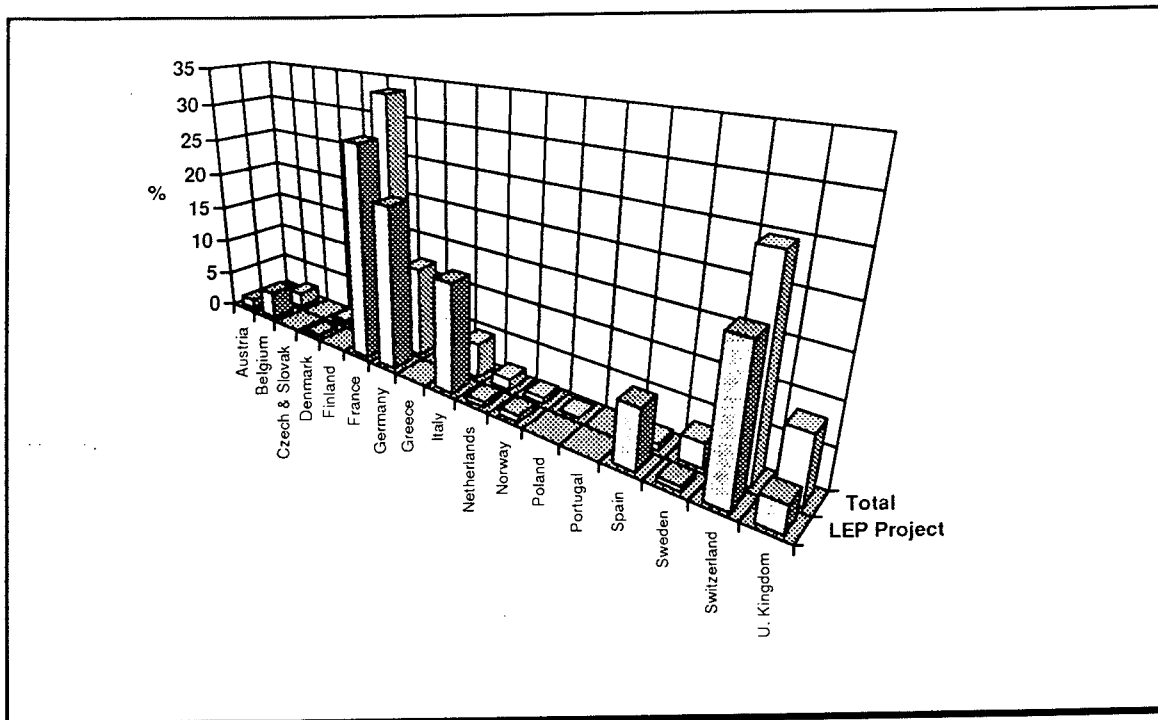


Figure A.6. Total distribution of purchases and for the LEP project only.

We see that for the LEP project the purchases distribution is more aderenth to the industrialization level of different countries. Countries like Germany, Italy, Spain and Belgium shows infact high percentages of purchases. Appendix A contains a list including the names of the companies, the subjects and the amounts of major conctracts stipulated for LEP realization.

A.6. Accelerators.

The best way to understand the intricate set of CERN's accelerators, that constitute today the biggest research facility in the field of particle phisics, is to follow the historical steps that conduced to the actual configuration. We need to look back to the

Europe of the early 1950s, when a far-sighted group of scientists and politicians envisioned a new adventure in science - a European scientific laboratory.

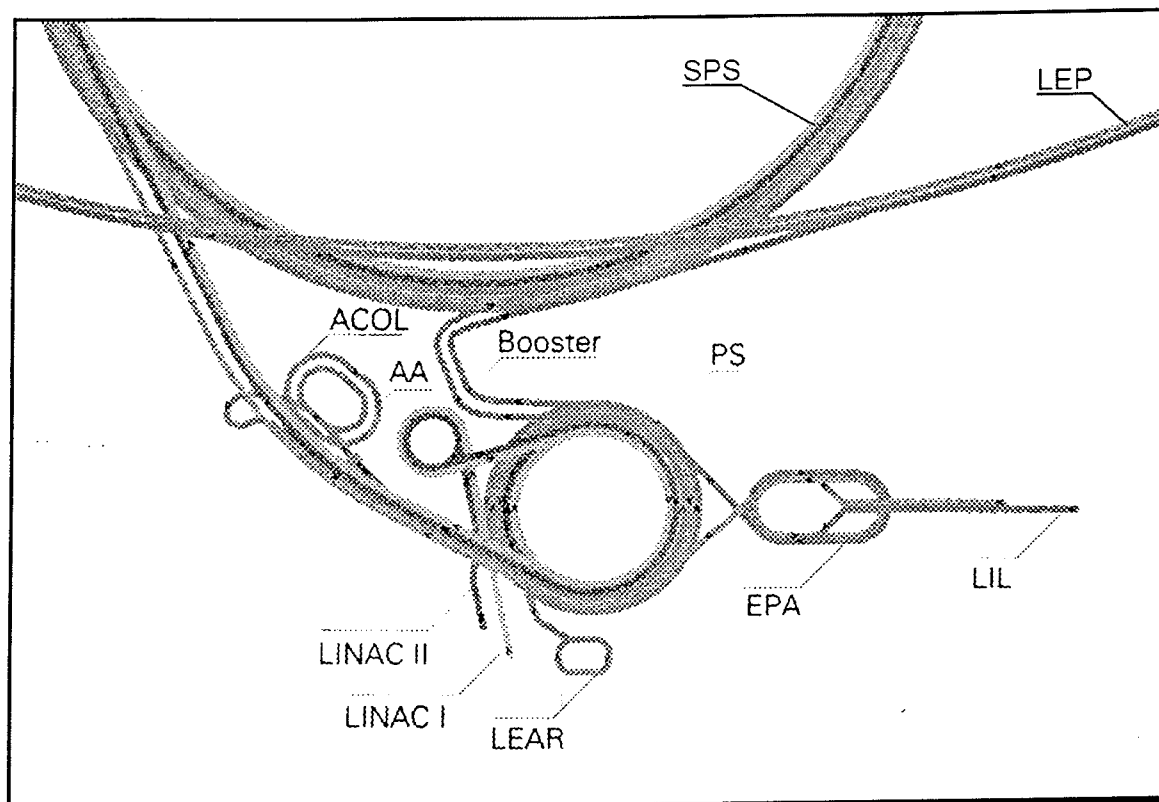


Figure A.7. The CERN's accelerators complex.

Even then, it was clear that state-of-the-art science needed research facilities larger and more complex than individual nations could afford. In this way Europe's leading role in fundamental science, so evident before 1939, would be restored, at the same time bringing together people from countries which had been at war only a few years before.

By 1954, enough countries had agreed to the idea for work to begin on the undeveloped Meyrin site in Switzerland on a particle accelerator - a synchro-cyclotron to accelerate protons to 600 MeV. This was not an ambitious project, even by the

modest standards of the time, but it got a research programme underway quickly and provided valuable experience.

In parallel, CERN began to build a very powerful machine - the Proton Synchrotron (PS). This came into operation in November 1959 and for a time was the most powerful particle accelerator in the world (supplying experiments with 28 GeV beams of protons). Over thirty years later, this machine remains the kingpin of CERN's unique system of interconnected accelerators. However it has long since stopped providing particle beams directly to experiments. Most of the time it gives the particles (protons, antiprotons, electrons, positrons, nuclei) their first big boost in energy before feeding them to the newer and larger machines, where they are taken to even higher energies.

From 1971 to 1984, CERN used a new type of machine - the Intersecting Storage Rings (ISR) - built on an extension of the Meyrin site into neighbouring French territory. Beams of protons previously accelerated in the PS were stored in two intertwined rings and smashed together. As well as opening up a new frontier of physics, the ISR provided scientists with valuable experience in building and operating this kind of machine, showing in particular that big detectors should surround the collision point to intercept all the emerging debris.

To answer the questions which resulted, work began in the early seventies on a big new machine, the SPS Super Proton Synchrotron. This produced its first 400 GeV proton beams in 1976, and was later uprated to 450 GeV. To house this seven kilometre ring, the CERN site was extended further into France, the protons crossing and recrossing the international frontier on each SPS circuit.

Even while the SPS was getting into its stride, a bold proposal was tabled to use it as a particle collider. To avoid having to build a second ring, the idea was to accelerate protons and their antimatter counterparts (antiprotons) in opposite directions in the same ring and make them collide. Antiprotons are rare - one is produced haphazardly for every hundred thousand or so protons hitting a target, but physics required about a billion of them at once.

The seemingly impossible was achieved by a new technique - 'stochastic cooling' - to control the precious anti-protons and shape them into an intense beam. In 1983, the idea paid off with the discovery of the W and Z particles - the carriers of the weak nuclear force - confirming an elegant theory uniting the description of this force with the everyday picture of electricity and magnetism.

This combined 'electroweak' theory is one of the great achievements of twentieth century physics. The discovery of the W and Z particles by Carlo Rubbia's team and the development of stochastic cooling by Simon van der Meer earned them the Nobel Prize for Physics in 1984. Science at this level needs foresight and planning.

Even while the SPS proton-antiproton collider was inching its way towards its historic discovery, the plans for the LEP (Large Electron Positron) machine were being finalized, and civil engineering began after a groundbreaking ceremony in September 1983 attended by the Presidents of CERN's two host nations - Francois Mitterrand of France and Pierre Aubert of Switzerland. LEP began operations in July 1989.

For the future, CERN is considering the LHC (Large Hadron Collider) project, installing a ring of powerful superconducting magnets in the LEP tunnel to collide proton beams of up to 8 TeV (8000GeV).

The names, the type of particles accelerated, and the power of the accelerators are summarized down here:

NAME	ENERGY	DIAMETER /DEPTH	COMMENTS
ISOLDE (Isotope Separators On Line)	1 GeV	---	Facility served by the Booster Proton-Synchrotron to study nuclei far from stability.
PS (Proton Synchrotron)	28 GeV	200 m. surface hall	Able to accelerate several types of particles simultaneously, and to decelerate antiprotons. Today mainly used as a pre-accelerator for protons, antiprotons and heavy ions for the SPS, and for leptons for LEP.
ISR (Intersecting Storage Ring)	---	---	World's first proton collider, closed in 1984 for financial reasons.
SPS (Super Proton-Synchrotron)	315+315 GeV	2.2 km. 23-65m. underground	Initially designed as a conventional accelerator, today also operates as a collider for protons and antiprotons.
AAC (Antiproton Accumulator Collector)	3.5 GeV	50m. surface hall	A facility supplying antiproton beams to SPS and LEAR experiments.
LEAR (Low Energy Antiproton Ring)	2 to 0.1 GeV	25 m. surface hall	This unique machine decelerates antiprotons for the study of antimatter at low energies.
LIL (Linear Injector for LEP)	0.6 GeV	100 m. long	The source of electrons and positons for LEP.
LEP (Large Electron Positron Collider)	55 + 55 GeV	8.6 km. 50-150 m. underground	The world's largest collider. An increase in energy will be achieved by gradual installation of a superconducting accelerator system.

Table A.1. CERN's accelerators.

Appendix B : Synthesis of the "Designing Graphical User Interfaces" tutorial.

Appendix B. Basic principles of Visual Communication.

Table of Contents.

B. Basic Principles of Visual Communication.	II
B.1. Organize.	III
B.1.1. Consistency.	III
B.1.2. Screen Layout.....	V
B.1.3. Navigability.	VI
B.2. Economize.	VII
B.2.1. Simplicity.....	VII
B.2.2. Clarity.	VIII
B.2.3. Distinctiveness.	IX
B.2.4. Emphasis.	X
B.3. Communicate.....	XI
B.3.1. Legibility.	XI
B.3.2. Readability.....	XII
B.3.3. Typography.	XIII
B.3.4. Symbolism.	XIV
B.3.5. Multiple Views.	XV
B.3.6. Color.	XV
B.4. Color Design Principles.....	XVII
B.4.1. Color organization.	XVII
B.4.2. Color Economy.	XVIII
B.4.3. Color Emphasis.	XVIII
B.4.4. Color Communication.....	XIX
B.5.5. Color Interactions.....	XX
B.4.6. Color Symbolism.	XX

B. Basic Principles of Visual Communication.

Researchers and developers need a high-quality, integrated approach to the graphical presentation of all elements of GUI design. Principles from information oriented, systematic design can provide guidance for simple, clear, consistent solutions to the design of menus, windows, icons, dialogue boxes, and control panels that are not explicitly specified by current GUI paradigms.

A GUI design must account for the following :

- A comprehensible mental image (metaphor).
- Appropriate organization of data, functions, tasks, and roles (cognitive model).
- Efficient navigation schema among these data and functions, tasks, and roles.
- Quality appearance characteristics (the look).
- Effective interaction sequencing (the feel).

Graphic design can help GUIs achieve their potential to communicate. Information-oriented, systematic graphic design is the use of typography, symbols, color, and other static and dynamic graphics to convey facts, concepts, and emotions. Graphic design is found in almost every facet of life, from the design of books, films, magazines, and videos, to the design of exhibits, highway signs, and maps. Information-oriented, systematic graphic design helps people understand complex information.

The manipulation of the visible language is a basic task of GUI design. The concept of visible language refers to all the graphical techniques used to communicate the message or content. The term includes the following:

- Layout: formats, proportions, and grids; 2-D and 3-D organization.
- Typography: selection of typefaces and typesetting, including variable-width sans serif and serif fonts, as well as fixed-width fonts found on most computer displays.
- Color and texture: color, texture and light that convey complex information and pictorial reality.
- Imagery: sign, icons, and symbols, from the photographically real to the abstract.

-Animation: a dynamic or kinetic display that is especially important for video-related imagery.

-Visual identity: the additional, unique rules that lend overall consistency to a user interface.

A set of principles can be useful to improve visual communication in a user interface.

This can be summarized in:

-Organize: Provide the user with a clear and consistent conceptual structure.

-Economize: Maximize the effectiveness of a minimal set of cues.

-Communicate: Match the presentation to the capabilities of the user.

B.1. Organize.

There are several important concepts within the major principle of organization: consistency, screen layout, relationships, and navigability.

B.1.1. Consistency.

There are four aspects of consistency: internal consistency, external consistency, real world consistency, and when not to be consistent; that is, when to deviate from the norm.

The principle of internal consistency suggest to observe the same conventions and rules for all elements of the GUI. Figure 4.1. provide an example where the same kind of elements are shown in the same place, and those with different kinds of behaviour each have their own special appearance. Casual differences should be avoided. Without a strong motivating reason, these casual differences cause the user to work harder to understand the essential message of the display.

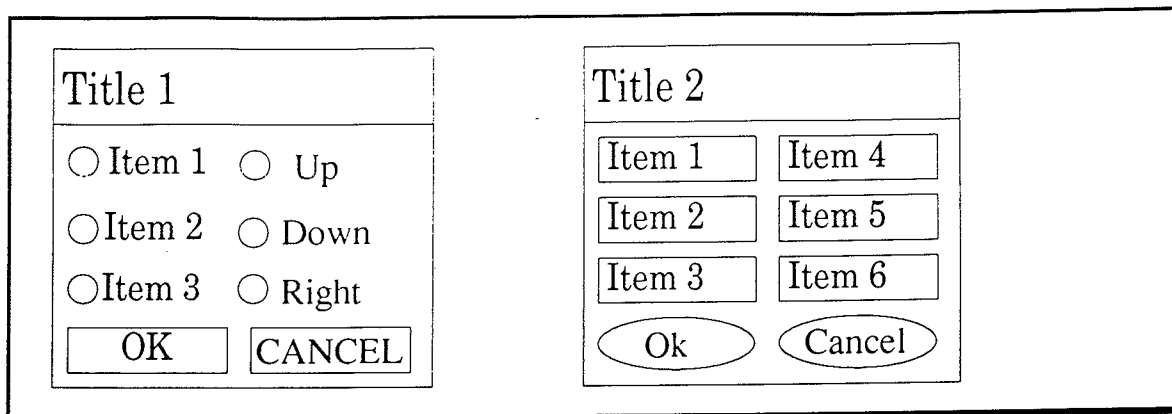


Figure B.1. Internal Consistency - Dialogue Boxes.

The second point is external consistency: follow existing platform and cultural conventions across user interfaces. Figure B.2. provides an example. The tool icons in the example comes from different graphical applications (PaintbrushTM, DesignerTM, and Corel DrawTM) but in all of them they conserve the same meaning. The lack of external consistency can cause confusion for users moving across applications.

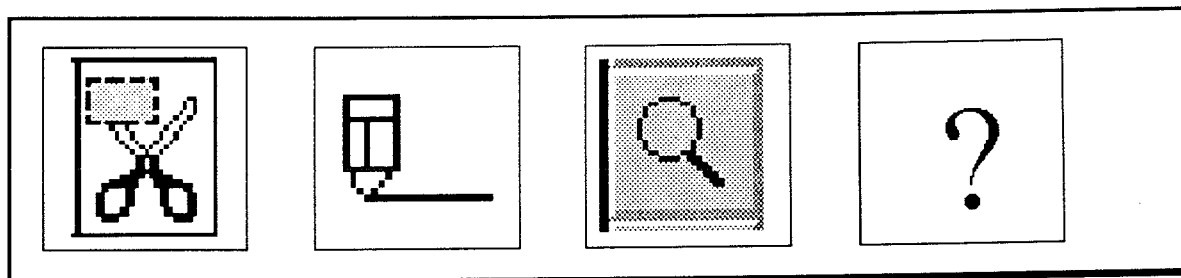


Figure B.2. External Consistency for Text Tool Icons.

The third point is real-world consistency: make the conventions consistent with real-world experience. Figure B.3. provides an example. It is possible to take advantage of objects, processes, or events that are already familiar to the user and build upon this knowledge. This principle is the basis of the success of the desktop metaphor.

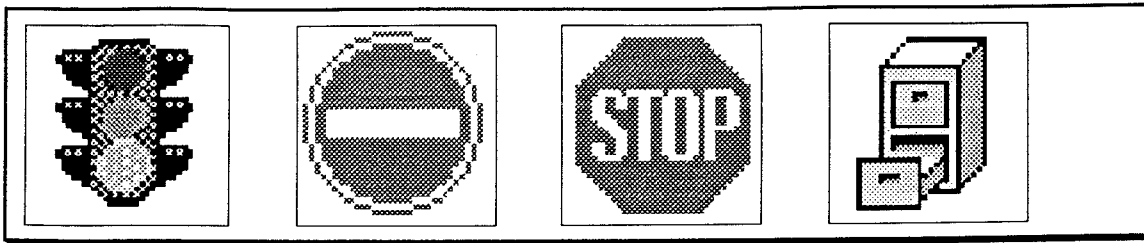


Figure B.3. Real-World Consistency.

The fourth and last point is innovation: deviate from existing conventions only when doing so provides a clear benefit to the user. In other words, there should be a good reason for being inconsistent.

B.1.2. Screen Layout.

The second major way to organize visual displays is to design their spatial layout or composition. There are three primary ways of achieving organization of screen layout: use a grid structure, standardize the screen layout, and group related elements.

Grids of horizontal and vertical lines can help locate items in menus, major window components, and other twodimensional compositions such as dialogue boxes or control panels. The exact measurements must be worked out very carefully to take into account the content to be displayed, spaces between text groups, the screen resolution, and the viewing conditions. Generally, the maximum number of major divisions in the horizontal or vertical direction follows what the human factors specialists call Miller's magic number, namely, 7 ± 2 . This recommendation derives from the maximum number of coding items we can easily retain in short-term memory.

The same idea can be applied to a control panel, menu, or dialogue box, or even the small area of icons, in which case a small grid helps locate individual pieces of these pictograms and ideograms. Another technique helpful in achieving visual organization is to establish clear relationship by linking related elements and disassociating unrelated elements. This simple precept may be complex to carry out in practice. Figure B.4. presents an example where at the left, shape, location, and value can create strong

visual relationship that may be completely inappropriate. In the improved example on the right, the relationships are clear, consistent, appropriate and strong.

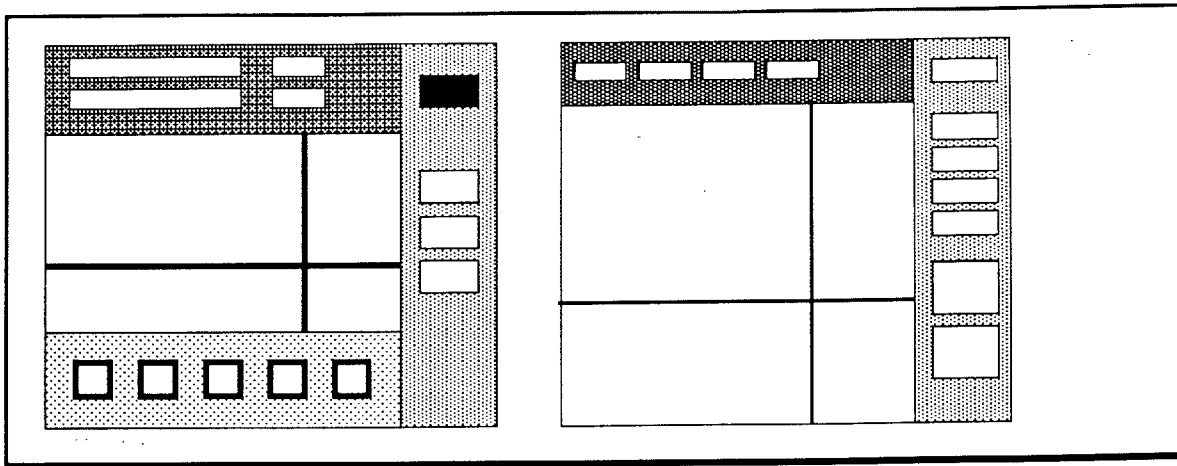


Figure B.4. Relationships.

B.1.3. Navigability.

The final means of organizing visual elements is navigability. Important techniques include to provide an initial focus for the viewer's attention, to direct attention to important, secondary, or peripheral items, and assist in navigation throughout the material. Figure B.5. provide an example of poor design and an example of improved design. It is important to notice, in this example, how very simple techniques of spatial layout and color (in this case, gray levels) can help focus the viewer's attention to the most important titlebar areas of the display. Then, the bulleted items guide the viewer through the secondary contents.

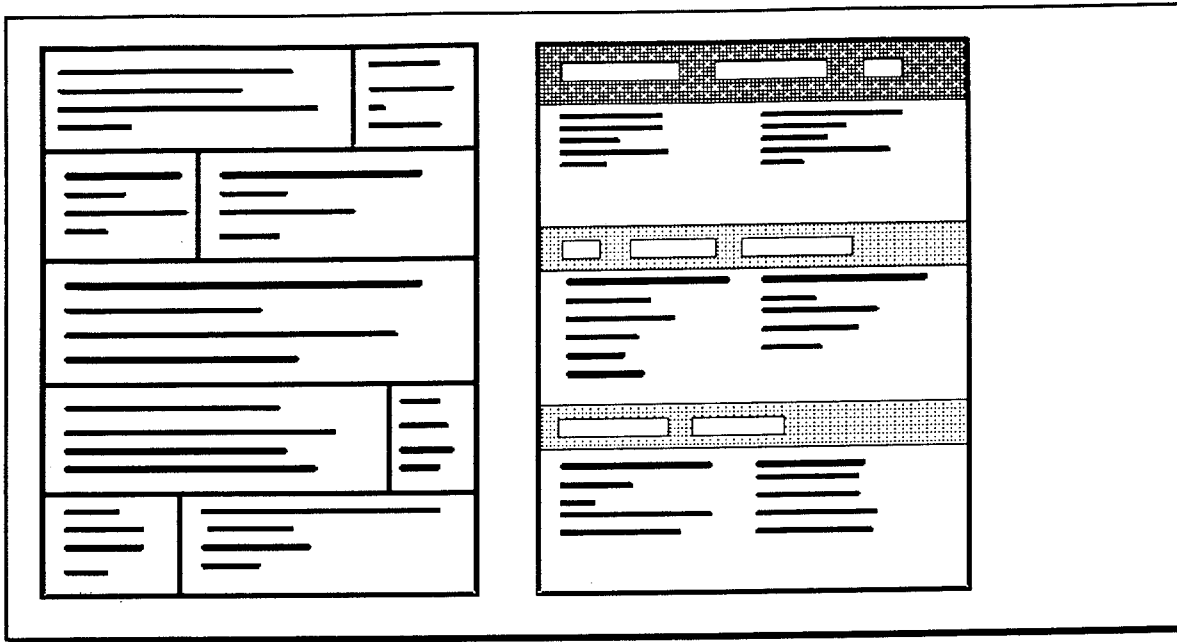


Figure B.5. A Navigation Example.

B.2. Economize.

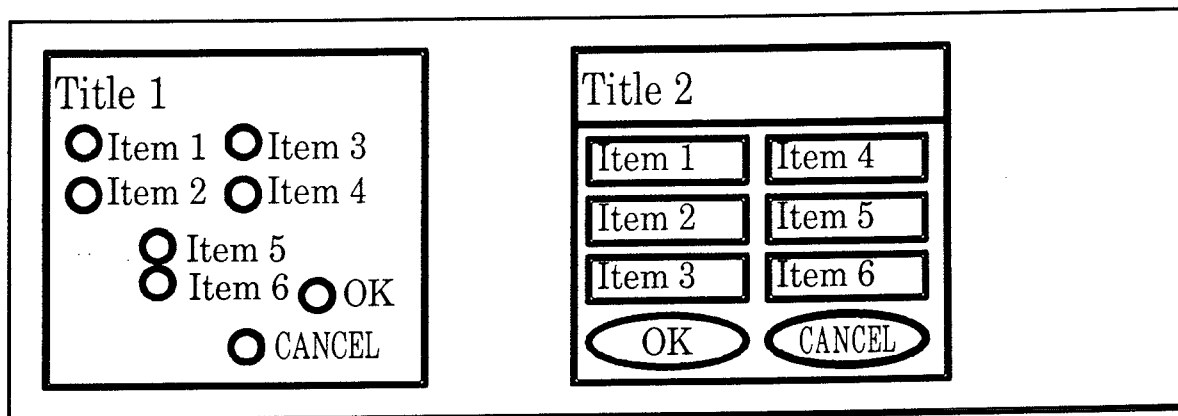
The second major principle is to economize (doing the most with the least). The basis is that, the fewer are the controls, the easier it looks to use and the easier it is to find the relevant controls. So we should really investigate on the number of controls does a device need and minimize them in order to make it look easy.

The concept of economy can be broken down into four major subtopics: simplicity, clarity, distinctiveness, and emphasis.

B.2.1. Simplicity.

Simplicity suggests that only those elements that are essential for communication should be included in the control panel. In addition, we should be as unobtrusive as possible. We should not assume that users are stupid; everyone has natural limits to the rate at which he can take in information-rich content. If we exceed natural limits, we will induce indifference, boredom, or anxiety. No matter to whom we are

communicating, simplicity is a virtue, unless our objective is to confuse, entertain, or beautify, in which cases, over indulgence may be justified. In general, the GUI components should be modest and not overly attract the viewer's attention. Users should be almost unconscious of the GUI working to convey meaning. Figure B.6. provides an example.



FigureB.6. Complicated and Simpler Design.

In the example on the left, the design is overdone, with useless differentiation. At the right, the presentation is simpler, with fewer extraneous changes of direction, typefaces, and capitalization.

B.2.2. Clarity.

The second technique for being economical is clarity: design all components so that their meaning is not ambiguous. Figure B.7. provides an example.

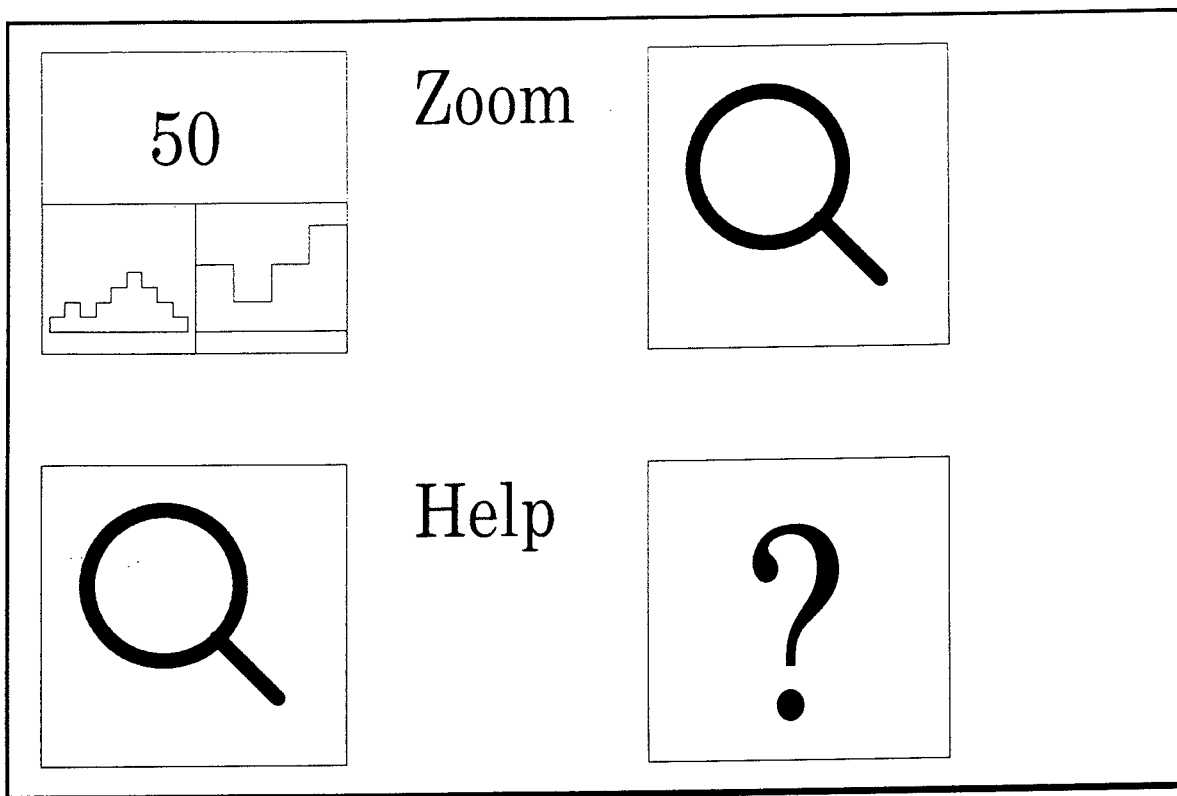


Figure B.7. Ambiguous and Clear Icons.

In this examples of icons, the ones on the left can be confusing, while the ones on the right are more understandable. The same symbol appears in two situations.

B.2.3. Distinctiveness.

The third technique for achieving economy is distinctiveness: distinguish important properties of essential elements. Figure B.8. illustrates this technique.

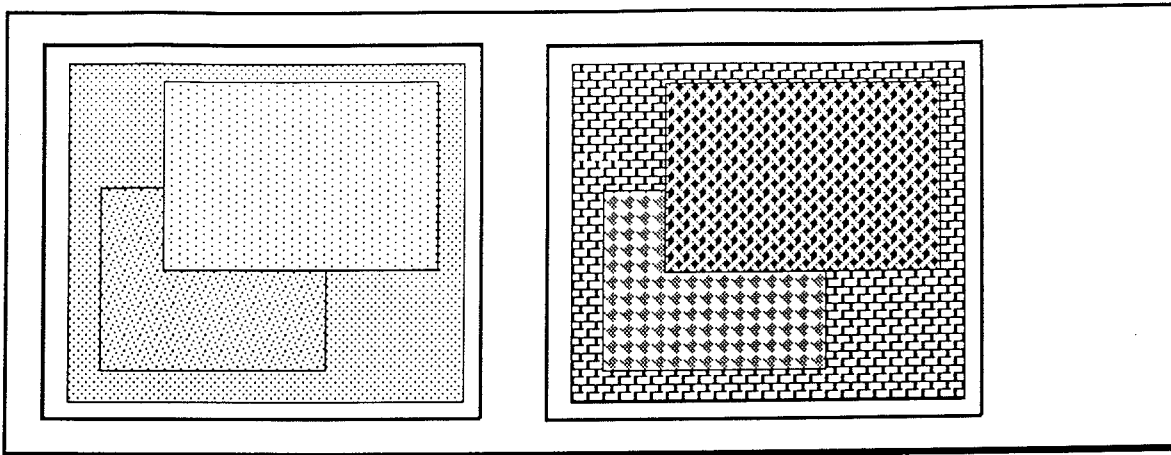


Figure B.8. Too Little and Too Much Distinctiveness.

The first example shows a typical screen design in which everything is too bland, too much the same. The second example shows an approach in which every component is trying to be too distinctive. A happy medium is the correct solution.

B.2.4. Emphasis.

The final technique of economizing is emphasis. In general, make the most important elements salient, that is, easily perceived. De-emphasize non-critical elements, and minimize clutter so that critical information is not hidden. Figure B.9. provides an example.

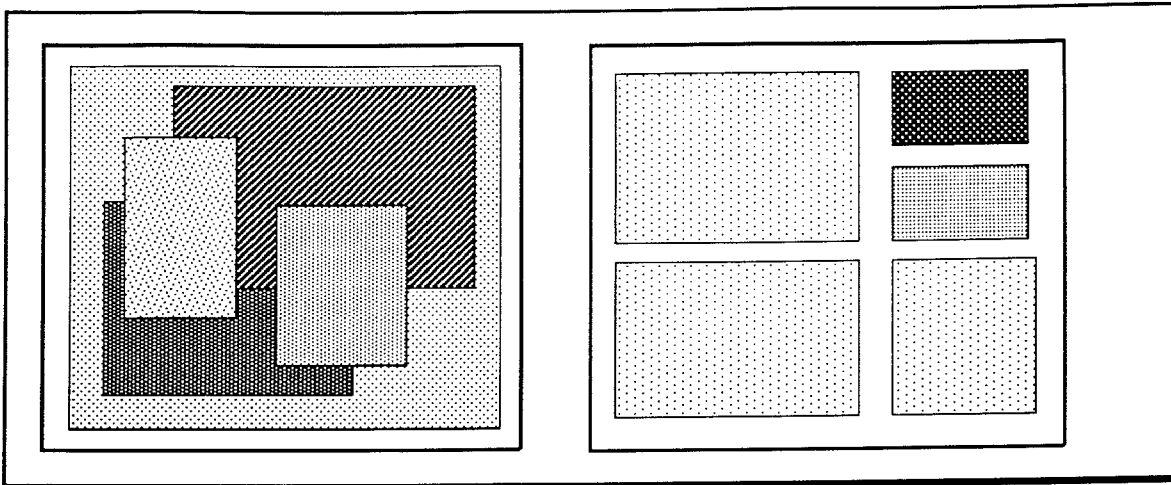


Figure B.9. Too Much and Improved Emphasis.

The first figure is busy and emphasizes too many things at once, making a clutter. The second example establishes a clear hierarchy. This approach can be used in achromatic (black, gray, white) as well as polychrome displays.

B.3. Communicate

The previous discussions have explored the principles of organization and economy. The third main principle is to communicate. To communicate successfully, the GUI must keep in balance these factors: legibility, readability, typography, symbolism, multiple views, and color.

B.3.1. Legibility.

The first factor is legibility: design individual characters, symbols, and graphic elements to be easily noticeable and distinguishable. Figure B.10. shows some examples.

Type fonts, desktop icons, control panel symbols, and so on, all need to be designed so that their parts show up well. You must select visualization techniques that are appropriate to the output display technology (for example, the spatial resolution, color, and animation characteristics). Another factor to remember is that dark screen backgrounds in brightly lighted rooms may cause distracting reflections that can

diminish screen legibility. In contrast, brightly lighted screens in dark rooms may be too glaring and hard to see.

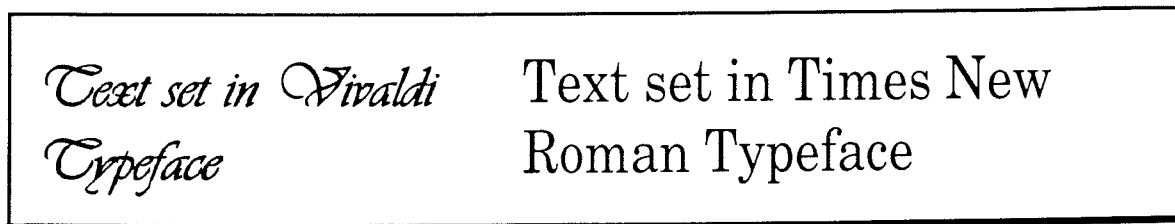


Figure B.10. Illegible and Legible Text.

The top left example is a classic case of an illegible font. The example on the right is much better.

B.3.2. Readability.

Closely related to legibility is readability. The term readability means that the display is comprehensible, that is, easy to identify and interpret, as well as inviting and attractive.

Figure B.11. presents an example.

Unreadable: Design components to be
easy to interpret and understand. Design
components to be inviting
and attractive.

Readable

Design components to be easy to interpret and
understand.

Design components to be inviting and attractive.

Figure B.11. Unreadable and Readable Text.

The top example shows a more unreadable text in which it is difficult to detect the presence of a subtitle. The second example shows a more readable arrangement of the same contents.

B.3.3. Typography.

One of the key contributors to legibility and readability is the use of typography in the user interface. We still rely on text, tables, lists, and annotation, even if we are showing charts, diagrams, or other graphical displays. Typography includes the characteristics of individual elements (typefaces and typestyles) and their groupings (typesetting techniques). Use a small number of typefaces of suitable legibility, clarity,

and distinctiveness to distinguish between different classes of information. Within each typeface, select a set of enhanced letterforms, punctuation marks, and symbols. Within menus, dialogue boxes, control panels, forms, and other window components, the point size, word spacing, paragraph indentation, and line spacing should be adjusted to enhance the readability and to emphasize critical information. The following are some basic recommendations:

- Generally, a maximum of three typefaces in a maximum of three point sizes should be used, no matter what the application.
- Lines of text should have from 40 to 60 characters maximum, and words should be spaced correctly, usually the width of a lower case "r" or "i" for a variable-width text.
- Text should be set in appropriate formats, that is, text flush left, numbers flush right, avoid centered text in lists, and avoid short justified lines of text.
- For fixed-width fonts, justified lines of text can slow reading speed by 12 percent.
- Use uppercase and lowercase characters whenever possible, that is, avoid all capital lines of text, which can also slow reading speed by 12 percent.

Of course, one can always find exceptions to these guidelines, but the basic recommendations should serve for most typographic settings.

B.3.4. Symbolism.

Typographic characters are not the only visible language elements that must be designed so that they communicate their content efficiently. Icons, symbols, charts, maps, diagrams, and photographic imagery used in the displays also must be carefully selected and refined to communicate the desired contents. The specific principles for designing charts, maps diagrams, pictograms, and ideograms are more numerous than space permit at this time.

B.3.5. Multiple Views.

One important technique for improving communication in the user interface is to provide multiple perspectives on the display of complex structures and processes. An important principle of good design, then, is making use of these multiple views:

- Multiple forms of representation.
- Multiple levels of abstraction.
- Simultaneous alternative views.
- Links across references.
- Metadata, metatext, metagraphics.

B.3.6. Color.

Color discussions can be confusing because scientists, artists, designers, programmers, and marketing professionals describe color phenomena in different ways. Let's first discuss the color terms using one useful set of dimensions: hue, value, and chroma.

These color terms are derived from the Munsell system of color used by artists, designers, and manufacturers. The dimensions differ from the red, green, and blue basis of the RGB color system familiar to users of CRT display devices. The approach also differs from another color system, the International Commission on Illumination (CIE), which uses specific spectral measurements to unambiguously specify color.

Hue is the spectral wavelength composition of a color that produces perceptions of being blue, orange, green, and so on. Value is the relative amount of lightness or darkness of the color in a range of black to white (also called intensity). Chroma is the purity of the color in a scale from gray to the most vivid variant of the perceived color (also called saturation). Brightness, a fourth key term, refers to the amount of light energy creating the color.

One important point to bear in mind is that color combinations can take place in different media in several ways. CRT screens typically combine light additively, while most hardcopy color is obtained through subtractive mixtures of pigments, dyes, or tints. Typical four-color process printing of images on paper uses closely spaced small

dots of cyan, yellow, and magenta, plus black (or CYMK, the graphic arts industry's specification of color) to achieve an optical mixing of colors due to the eye's inability to resolve the individual color elements in "full-color" graphic arts printing.

A typical need for screen design is to combine colors so they make visual sense.

Beyond the basic technology of color for screen display, it is important to understand how color can help communication. Here are some basic advantages:

- Emphasize important information.
- Identify subsystems or structures.
- Portray natural objects in a realistic manner.
- Portray time and progress.
- Reduce errors of interpretation.
- Add coding dimensions.
- Increase comprehensibility.
- Increase believability and appeal.

With respect to learning and comprehension, color is superior to black-and-white in terms of the viewer's processing time and emotional reactions, and there is a difference in a viewer's ability to interpret information. People often can learn more from a color display if color is used correctly. Another crucial factor is that color is more enjoyable. With respect to memory performance, memory for color information appears to be superior to black-and-white.

These benefits indicate that color can work for you in communicating facts, concepts, and emotions; but color is definitely a complex tool. At the same time, color has these potential drawbacks:

- Color requires more expensive and complicated display equipment.
- Color displays may not accommodate color deficient vision, which occurs in about 8 percent of Caucasian males.
- Some colors may cause visual discomfort and afterimages.
- Color may contribute to visual confusion or may catalyze negative associations through cross-disciplinary and cross-cultural connotations.

Because color can be misused easily there is a clear need for guidelines. Establishing complete rules or specifications for color use is difficult. Anyway the International Standards Organization (ISO) has begun by publishing draft standards for some aspects of basic color legibility. The complexity of the problem justify a chapter describing the basis of color design.

B.4. Color Design Principles.

To keep recommendations simple, we can, as in the previous chapter, refer to basic principles, but this time applied to color. We'll examine selected guidelines under the following principles: color organization, color economy, color emphasis, color communication, color interaction and color symbolism.

B.4.1. Color organization.

The first set of recommendations about color concerns consistency of organization. Use color to group related items, and use a consistent color code for screen displays, documentation, and training materials. In general, similar colors should imply a relation among objects. A viewer can sense the relatedness by color over space and over time in sequences of images. Be complete and consistent in your color groupings. For example, command and control colors in menus should be avoided for information coding within a work area, unless a specific connection is intended.

Another recommendation is to use similar background colors for related areas. This color coding can subtly orient the viewer to the conceptual link among these areas. Once color coding is established, the same colors should be used throughout the GUI and all related publications. This color continuity may require designing colors to appear in different media. Remember: CRT screens use additive color mixtures, while most hardcopy devices use subtractive color mixtures. The color gamuts (that is, available color ranges) of these two media usually are not identical.

B.4.2. Color Economy.

The next recommendations concern economical use of color. The principle of color simplicity suggests using a maximum of 5 ± 2 colors where the meaning must be remembered. Note that this maximum is even less than Miller's magic number from human factors, 7 ± 2 . If appropriate, use redundant coding based on shape, as well as color. The basic idea is to use color to enhance black-and-white information, that is, design the display to work well first in black-and-white. For documentation tasks, for example, using color to portray an object, the maximum number of colors needed is dependent on the application. For aesthetic purposes such as design style, emotional expression, or realism, many more colors may be required. To code a large set of colors, use the spectral sequence: red, orange, yellow, green, blue, and violet. Francine Frome, a human factors researcher, has shown that viewers see a spectral order as a natural one and would select red, green, and blue as intuitive choices for the front, middle, and back layers, respectively, when viewing a multilayer display. Note, however, that brightness can change a viewer's perception of depth. If the colors are balanced, then red seems to come forward. We should also take care to use redundant coding of shape as well as color, which aids those with color-deficient vision and makes the display more resilient to color distortions caused by ambient light changes or by medium-to-medium conversion. Ambient light can cause changes in perceived hue, value, and chroma. Conversion from one medium to another can cause unforeseen and sometimes uncontrollable changes. Remember that among Caucasian viewers, 8 percent of males have some form of color-deficient viewing.

B.4.3. Color Emphasis.

Among the remaining principles, color emphasis suggests using strong contrast in value and chroma to focus the user's attention on critical information. The use of bright colors for danger signals, attention getters, reminders, and cursors is entirely appropriate.

High chroma red-alerts seem to aid faster response than yellow or yellow-orange if brightness is equal, but of course this also depends upon the background colors. When too many figures or background fields compete for the viewer's attention, then confusion arises. The hierarchy of highlighted, neutral, and lowlighted states for all areas of the visual display must be carefully designed to maximize simplicity and clarity. Here again we find the basic maxim: simplicity, clarity, and consistency are especially important for color design.

Use saturated or high-chroma colors for older viewers, or for those who have viewed displays for very long periods of time. Bear in mind that frequent, but short-term viewing can benefit from low-chroma displays, and that very bright displays of letters, symbols, or lines tend to bloom, that is, the light spreads out against the background.

B.4.4. Color Communication.

In color communication, recommendations for legibility include the following:

- Use colors appropriately for the central and peripheral areas of the visual field.
- Use color combinations whose discriminability is influenced least by the relative area of each color.
- Do not use colors that are simultaneously high in chroma and spectrally extreme.

The outer edges of the retina are not particularly sensitive to colors in general. Thus, red or green should be used in the center of the visual field, not in the periphery. If they are used at the periphery, some signal to the viewer must be given to capture attention, for example, size change or blinking. Use blue, black, white, and yellow near the periphery of the visual field. The retina remains sensitive to these colors near the periphery. Use blue for large areas, not for text type, thin lines, or small shapes. Blue-sensitive color receptors are the least numerous in the retina (approximately 5 percent), and are especially infrequent in the eye's central focusing area, the fovea. Blue is good for screen backgrounds.

If the colors change in size in the imagery, you should bear in mind the following: As color areas decrease in size they appear to change their value and chroma. Use colors

that differ both in chroma and value (lightness). Do not use adjacent colors that differ only in the amount of pure blues, because the edge between the two colors will appear to be fuzzy. To avoid this you might use a dark blue and a light blue. Regarding use of simultaneously high-chroma, spectrally extreme colors, you can see that strung contrasts of red/green, blue/yellow, green/blue, and red/blue can create vibrations, illusions of shadows, and afterimages. Unless you need to create a special visual effect, avoid these combinations.

In general, use light text, thin lines, and small shapes (white, yellow, or red) on medium to dark backgrounds (blue, green, or dark gray) for dark viewing situations. Typical low ambient-light viewing situations are those for slide presentations, workstations, and video. Video displays produce colors that are lower chroma. Use dark (blue or black) text, thin lines, and small shapes on light (light yellow, magenta, blue, or white) backgrounds for light-viewing situations. Typical viewing situations are those for overhead transparencies and paper. Reserve the highest contrast in figure-field relationships for text type.

B.5.5. Color Interactions.

The interaction of color is a very complex subject that cannot be presented in this limited space. Designers of users interfaces need to become familiar with these phenomena, which students of art and design often study. We can simply say that color interaction with the background color fields can create illusions like, for example, one color that appears in two and two colors that appears in one.

B.4.6. Color Symbolism.

Finally, we recall the importance of symbolism in communication. The suggestion is to use color codes that respect existing cultural and professional usage. We should use color connotations with great care. Connotations vary strongly among different kinds of viewers, especially from different cultures. The correct use of color requires careful analysis of the experience and expectations of the viewers.

For example, mailboxes are blue in the USA, bright red in England, and bright yellow in Greece. If color is used in an electronic mail icon on the screen, this suggests that color sets might be changed for different countries to allow for differences in international markets.

**Appendix C : The listing of the POWP.C equipment
module and the HWACCESS.C functions.**

Appendix C : Listings.

Table of contents.

The POWP.C equipment module.	I
The HWACCESS.C functions.	VI

The POWP.C equipment module.

```
/* Program POWP.C for an equipment module written in Quicke C */
/* POWP is built for Quadrapole */
```

```
#include "stdlib.h"
#include "I:\Necnod\nodal.h"
```

```
typedef struct
{
    real MINV;real MAXV;real SCLR;real SCCR;real SCLW;real SCCW;
    int ADRR;int ADRW;int CHNR;int CHNW;int ADRD;long CTRW;
    int CHNIO;int CHNIF;int CHNIS;int CHNIR1;int CHNIR2;int CHNII;int CHNIE;
    int CHNOO;int CHNOF;int CHNOS;int CHNOR;real CCV; int STCC;
} POW_DESC;
```

```
static POW_DESC POWP_Table[MAX_POWP];
```

```
#define WRITE -1
#define READ 1
#define OFF 1
#define ON 2
#define STANDBY 4
#define INTERLOCK 8
#define LOCAL 16
#define FAULT 32
#define RESET 64
```

```
int waitms(int delay);
int OutBit(int BaseAddress, int Channel, int Bit);
int InBit(int BaseAddress, int Channel);
NodalError OutValue(int BaseAddress, int Channel, real SCL, real SCC,
real value);
NodalError InValue(int BaseAddress, int Channel, real SCL, real SCC,
real far *value);
```

```
NodalError D_POWP(real far *value, int rwflag, int eqno, char *property)
{
    NodalError cc;
    int eq,PI,AL,R1,R2,RF;
    real val;

    if ((eqno < 1) || (eqno > MAX_POWP))
    {
```

```

        printf("\nERROR: %d is not a legal equipment number\n", eqno);
        return illegal_equipment_number;
    }
    eq = eqno - 1;

    /* Property CCV Current Control Value */
    /* read value from Data Table */
    /* Write value to data table, scale using SCLW, write to hardware */
    if (!strcmp(property, "CCV"))
    {
        if (rwflag == WRITE) /* write value to hardware */
        {
            if (POWP_Table[eq].ADRW == -1)
            {
                printf("\nERROR: Equipment N. %d is not connected to
DAC\n", eqno);

                return device_not_connected;
            }
            if (*value < POWP_Table[eq].MINV || *value > POWP_Table[eq].MAXV)
            {
                printf("\nERROR: Equipment N. %d received %d as
CCV\n", eqno, *value);

                return out_of_range;
            }
            cc = OutValue(POWP_Table[eq].ADRW, POWP_Table[eq].CHNW,
POWP_Table[eq].SCLW, POWP_Table[eq].SCCW, *value);
            if (cc != 0) return cc;
            POWP_Table[eq].CCV = *value; /*UPDATE pow_tab */
            return 0;
        }
        if (rwflag == READ)
        {
            if (POWP_Table[eq].ADRW == -1)
            {
                printf("\nERROR: Equipment N. %d is not connected to
DAC\n", eqno);

                return device_not_connected;
            }

            *value = POWP_Table[eq].CCV;
            return 0;
        }
        printf("\nERROR: rwflag is equal to: %d\n", rwflag);
        return illegal_read_write;
    }
}

```

```

/*Property AQN Read current in Volts */
/*AIP is 1 ADC + Multiplexer; BaseAdres + 0 = Multiplexer chan
BaseAdres + 1 = Start Conversion....
*/
    if (!strcmp(property, "AQN"))
    {
        if (rwflag != READ) return illegal_read_write;
        if (POWP_Table[eq].ADDR == -1)
        {

```

```

        printf("\nERROR: Equipment N. %d is not connected to ADC\n",eqno);
        return device_not_connected;
    }
    cc = InValue (POWP_Table[eq].ADDR, POWP_Table[eq].CHNR,
POWP_Table[eq].SCLR, POWP_Table[eq].SCCR, value);
    if (cc != 0 ) return cc;
    return 0;
}

```

```

/* Property STAQ - Read/Back status of the power supply -
    Off = 0; On = 1; */
/* This register is set by the Hardware when something goes wrong!...*/

```

```

    if (!strcmp(property,"STAQ"))
    {
        if (rwflag != READ) return illegal_read_write;
        if (POWP_Table[eq].ADDRD == -1)
        {
            printf("\nERROR: Equipment N. %d is not connected to DIO\n",eqno);
            return device_not_connected;
        }
        AL = 0;
        if (InBit(POWP_Table[eq].ADDRD, POWP_Table[eq].CHNIO) == 1))
            AL |= ON;

        *value = AL;
        return 0;
    }

```

```

/* Property ADDR read or write the ADC BASE ADDRESS */

```

```

    if (!strcmp(property,"ADDR"))
    {
        if (rwflag == WRITE)
        {
            POWP_Table[eq].ADDR = *value;
            return 0;
        }
        if (rwflag == READ)
        {
            *value = POWP_Table[eq].ADDR ;
            return 0;
        }
        return illegal_read_write;
    }

```

```

/* Property ADRW read or write the DAC BASE ADDRESS */

```

```

    if (!strcmp(property,"ADRW"))
    {
        if (rwflag == WRITE)
        {
            POWP_Table[eq].ADRW = *value;
            return 0;
        }
        if (rwflag == READ)
        {

```

```

        *value = POWP_Table[eq].ADRW;
        return 0;
    }
    return illegal_read_write;
}

/* Property CHNR read or write the DAC ADC Channel */
if (!strcmp(property,"CHNR"))
{
    if (rwflag == WRITE)
    {
        POWP_Table[eq].CHNR = *value;
        return 0;
    }
    if (rwflag == READ)
    {
        *value = POWP_Table[eq].CHNR;
        return 0;
    }
    return illegal_read_write;
}

/* Property CHNW read or write the DAC Channel */
if (!strcmp(property,"CHNW"))
{
    if (rwflag == WRITE)
    {
        POWP_Table[eq].CHNW = *value;
        return 0;
    }
    if (rwflag == READ)
    {
        *value = POWP_Table[eq].CHNW;
        return 0;
    }
    return illegal_read_write;
}

/* Property CHNIO ON Read/Back Input Register Channel */
if (!strcmp(property,"CHNIO"))
{
    if (rwflag == WRITE)
    {
        POWP_Table[eq].CHNIO = *value;
        return 0;
    }
    if (rwflag == READ)
    {
        *value = POWP_Table[eq].CHNIO;
        return 0;
    }
    return illegal_read_write;
}

/* Property SCLW read or write the DAC scaling factor ("SCLWW") */
if (!strcmp(property,"SCLW"))
{

```

```

        if (rwflag == WRITE)
        {
            POWP_Table[cq].SCLW = *value;
            return 0;
        }
        if (rwflag == READ)
        {
            *value = POWP_Table[cq].SCLW;
            return 0;
        }
        return illegal_read_write;
    }

/* Property SCCW read or write the DAC scaling constant ("SCCW") */
if (!strcmp(property,"SCCW"))
{
    if (rwflag == WRITE)
    {
        POWP_Table[cq].SCCW = *value;
        return 0;
    }
    if (rwflag == READ)
    {
        *value = POWP_Table[cq].SCCW;
        return 0;
    }
    return illegal_read_write;
}

/* Property SCLR read or write the ADC scaling factor ("SCLRR") */
if (!strcmp(property,"SCLR"))
{
    if (rwflag == WRITE)
    {
        POWP_Table[cq].SCLR = *value;
        return 0;
    }
    if (rwflag == READ)
    {
        *value = POWP_Table[cq].SCLR;
        return 0;
    }
    return illegal_read_write;
}

/* Property SCCR read or write the ADC scaling constant ("SCCRR") */
if (!strcmp(property,"SCCR"))
{
    if (rwflag == WRITE)
    {
        POWP_Table[cq].SCCR = *value;
        return 0;
    }
    if (rwflag == READ)
    {
        *value = POWP_Table[cq].SCCR;
        return 0;
    }
}

```

```

    }
    return illegal_read_write;
}

/* Property MAXV read or write the MAXIMUM VALUE ("MAXV") */
if (!strcmp(property, "MAXV"))
{
    if (rwflag == WRITE)
    {
        POWP_Table[cq].MAXV = *value;
        return 0;
    }
    if (rwflag == READ)
    {
        *value = POWP_Table[cq].MAXV;
        return 0;
    }
    return illegal_read_write;
}

/* Property MINV read or write the MINIMUM VALUE ("MINV") */
if (!strcmp(property, "MINV"))
{
    if (rwflag == WRITE)
    {
        POWP_Table[cq].MINV = *value;
        return 0;
    }
    if (rwflag == READ)
    {
        *value = POWP_Table[cq].MINV;
        return 0;
    }
    return illegal_read_write;
}

return illegal_property;
}

```

The HWACCESS.C functions.

/* Common Library to Access HW from POWP and MPOWP */

```

#include "stdlib.h"
#include "I:\fecnod\nodal.h"
#include <conio.h>
#include <time.h>

```

```

int waitms(int delay)
{
    clock_t goal;

    goal = (clock_t)delay + clock();
    while( goal > clock() );
}

```

```

int InBit(int BaseAddress, int Channel)
{
    int Port, Shift, Value, Mask, Bit, Npar, Pgr, Offset, Cnfg;

    if( Channel <=0 || Channel > 98 ) return 0;

    if (Channel > 48) Channel = Channel - 2; /* Pins 49 and 50 are occupied by the +5VDC
supply and the ground */
    Npar = Channel % 2;
    Pgr = Channel/16 +1;
    if (Channel == 16*(Pgr-1)) Pgr = Pgr-1;

    switch(Pgr)
    {
        case 1:
            if (Npar == 0) /* Channel 2-->16 */
            {
                Offset = 0x06;
                Shift = 7 - ((Channel/2 - 1) % 8);
                Port = BaseAddress + Offset;
            }

            else if (Npar == 1) /* Channel 1-->15 */
            {
                Offset = 0x02;
                Shift = 7 - (Channel/2 % 8);
                Port = BaseAddress + Offset;
            }
            break;

        case 2:
            if (Npar == 0) /* Channel 18-->32 */
            {
                Offset = 0x05;
                Shift = 7 - ((Channel/2 - 1) % 8);
                Port = BaseAddress + Offset;
            }

            else if (Npar == 1) /* Channel 17-->31 */
            {
                Offset = 0x01;
                Shift = 7 - (Channel/2 % 8);
                Port = BaseAddress + Offset;
            }
            break;

        case 3:
            if (Npar == 0) /* Channel 34-->48 */
            {
                Offset = 0x04;
                Shift = 7 - ((Channel/2 - 1) % 8);
                Port = BaseAddress + Offset;
            }

            else if (Npar == 1) /* Channel 33-->47 */
            {

```

```

        Offset = 0x00;
        Shift = 7 - (Channel/2 % 8);
        Port = BaseAddress + Offset;
    }
    break;

case 4:
    if (Npar == 0) /* Channel 52-->66 */
    {
        Offset = 0x0E;
        Shift = 7 - ((Channel/2 - 1) % 8);
        Port = BaseAddress + Offset;
    }

    else if (Npar == 1) /* Channel 51-->65 */
    {
        Offset = 0x0A;
        Shift = 7 - (Channel/2 % 8);
        Port = BaseAddress + Offset;
    }
    break;

case 5:
    if (Npar == 0) /* Channel 68-->82 */
    {
        Offset = 0x0D;
        Shift = 7 - ((Channel/2 - 1) % 8);
        Port = BaseAddress + Offset;
    }

    else if (Npar == 1) /* Channel 67-->81 */
    {
        Offset = 0x09;
        Shift = 7 - (Channel/2 % 8);
        Port = BaseAddress + Offset;
    }
    break;

case 6:
    if (Npar == 0) /* Channel 84-->98 */
    {
        Offset = 0x0C;
        Shift = 7 - ((Channel/2 - 1) % 8);
        Port = BaseAddress + Offset;
    }

    else if (Npar == 1) /* Channel 83-->97 */
    {
        Offset = 0x08;
        Shift = 7 - (Channel/2 % 8);
        Port = BaseAddress + Offset;
    }
    break;
}

```

Value = inp(Port);

Value =~ (Value);/* This Line is done to take into account that the input is open collector*/

```

Mask = 1 << Shift;
Value = Value & Mask;
Bit =( Value >> Shift);
return(Bit);
}

NodalError OutValue(int BaseAddress, int Channel, real SCL, real SCC,
real value)
{
int pw,Low,High,Port;
NodalError cc;

if (cc=nfix(value * SCL + SCC, &pw)) return cc;
if ( pw >4095 ) pw=4095;
if ( pw < 0 ) pw=0;

High=pw>>8;
Low=pw & 0xff;
Port = BaseAddress + Channel * 2;
outp( Port , Low); /* CHANNEL 0 LOW BIT*/
outp( Port + 1 , High); /* CHANNEL 0 HIGH BIT */
inp ( BaseAddress + 15); /* UPDATE STROBE OUTPUT */
return 0;
}

NodalError InValue(int BaseAddress, int Channel, real SCL, real SCC, real far *value)
{
int Port;
int i;
int Low, High, Bits;
real val;
NodalError cc;

Port = BaseAddress;
outp (Port, Channel); /* CHANNEL CHNR SELECTED */
for (i = 0; i < 150; i++);
outp(Port + 1, 0); /* START CONVERSION */
i = 0;
do
{
i++;
High = inp (Port + 3); /*READ HIGH DATA */
if (i >= 10000)
{
printf("\n ERROR: Can not read from ADC(Base Address: %x, Channel:
%d)\n",BaseAddress,Channel);
return device_not_connected;
}
} while (High & 0x20); /* Bit 5 at 1 when the ADC is busy !!! */

Low = inp (Port + 2); /*READ LOW DATA */
Bits = ((High & 0xf) << 8) + Low;
val = ((Bits - SCC) / SCL);
i = (int) val; /* make integer to avoid numbers behind comma on console */
if ((val - i) >= .5) i++; /* GJF 930301 */
*value = (real) i;

```

```

        return ();

    }

/* error.h listing*/
#define MIN_DELAY 2000

extern char *erlst[];

/* Error Number Definitions*/
#define illegal_line      1
#define illegal_format    2
#define illegal_arithmetic 3
#define ambiguous         4
#define illegal_delimiter 5
#define zero_divide       6
#define WA_full           7
#define non_existent      8
#define wrong_type        9
#define link_exhausted    10
#define not_properly      11
#define un_allocated      12
#define no_such_line      13
#define illegal_shuffle   14
#define if_error          15
#define escape_typed      16
#define ask_error         18
#define argument_list_error 20
#define file_error        21

#define dimension_error    23
#define square_root_negative 24

#define log_negative       31 /* or = 0 */
#define device_not_connected 32
#define unauthorised_action 33
#define hardware_error     34
#define illegal_equipment_number 35
#define illegal_property   36
#define out_of_range       37
#define not_implemented    38
#define no_such_computer   39
#define string_filled      40
#define syntax_error       41
#define no_such_file       42
#define no_file_space      44
#define link_not_open      45
#define remitted_data_lost 46
#define end_of_file        47
#define def_area_full      52
#define def_syntax         53
#define illegal_string_set 54
#define serr1              54
#define string_function_failure 55
#define serr2              55
#define Sif_error          57 /* SERR4 on NORD */
#define Sask_error         58 /* SERR5 */

```

```

#define string_expected      59    /* SERR6 */
#define no_link              64
#define link_error           65
#define code_failure         66
#define bracket_missing      67
#define wrong_hexa           68
#define wrong_octal          69
#define illegal_character    70
#define illegal_operator     71
#define not_allowed          72
#define illegal_read_write   73
#define out_of_memory        74
#define database_not_loaded  75
#define illegal_command      76
#define resources_exhausted  77
#define file_not_open        78
#define camac_error          79    /*camac*/
#define no_such_loop         80    /*serial camac*/
#define simatic_error        81
#define driver_not_installed  82
#define port_not_available   83
#define scan_error           84    /*scanner*/
#define no_high_voltage      85    /*tape station*/
#define non_existent_clem    86
#define non_existent_fec     87
#define non_existent_client  88
#define cannot_lock          89
#define not_running          90
#define not_posted           91
#define not_accepted         92
#define dde_timeout          93
#define illegal_protocol     94
#define gpib_error           95
#define rs232_error          96
#define Fug_in_Transition    97
#define connection_timeout   98
#define illegal_mode         99
#define illegal_plsline      100
#define not_defined          101
#define net_write_error       102
#define invalid_data         103
#define timer_too_low        104
#define access_denied        105

```

Appendix D. Database files: EQPMOD.CSV,
FECADDR.CSV, EQPNAMES.CSV, and
POWP.CSV.

ExpNum	Name	FecName	MINV	MAXV	SCLR	\$CCR	SCLW	SCCW	ADRR	ADRW	CHNR	CHNW	CTRW	ADRD	CHNIO	CCV	AGN	STAG
2	GPS.QP040	POWER1	0	3500	1.17	0	1.17	0	1C0h	160h	2	2		100h	52	0	0	x
3	GPS.QP050	POWER1	0	2000	2.0475	0	2.0475	0	1C0h	160h	3	3		100h	52	0	0	x
4	GPS.QP170	POWER1	0	2000	2.0475	0	2.0475	0	1C0h	160h	4	4	1079bh	100h	54	0	0	x
5	GPS.QP180	POWER1	0	1250	3.2768	0	3.2768	0	1C0h	160h	5	5		100h	54	0	0	x
6	GPS.QP520	POWER1	0	3500	1.17	0	1.17	0	1C0h	160h	6	6		100h	54	0	0	x
7	GPS.QP530	POWER1	0	3500	1.17	0	1.17	0	1C0h	160h	7	7	1069bh	100h	56	0	0	x
8	GPS.QP540	POWER1	0	3500	1.17	0	1.17	0	1C0h	180h	8	8		100h	56	0	0	x
9	GLM.QP40	POWER1	0	3500	1.17	0	1.17	0	1C0h	180h	9	9		100h	56	0	0	x
10	GLM.QP50	POWER1	0	2000	2.0475	0	2.0475	0	1C0h	180h	10	10	1019bh	100h	58	0	0	x
11	GLM.QP60	POWER1	0	2000	2.0475	0	2.0475	0	1C0h	180h	11	11		100h	58	0	0	x
12	GHM.QP40	POWER1	0	3500	1.17	0	1.17	0	1C0h	180h	12	12		100h	58	0	0	x
13	GHM.QP50	POWER1	0	2000	2.0475	0	2.0475	0	1C0h	180h	13	13		100h	60	0	0	x
14	GHM.QP60	POWER1	0	2000	2.0475	0	2.0475	0	1C0h	180h	14	14		100h	60	0	0	x
1	CA0.QP40	POWER3	0	3500	1.17	0	1.17	0	1C4h	180h	0	0		100h	60	0	0	x
2	RA0.QP20	POWER3	0	3500	1.17	0	1.17	0	1C4h	180h	1	1		100h	60	0	0	x
3	RA0.QP30	POWER3	0	3500	1.17	0	1.17	0	1C4h	180h	2	2		100h	60	0	0	x
4	LA0.QP20	POWER3	0	3500	1.17	0	1.17	0	1C4h	180h	3	3		100h	60	0	0	x
5	LA0.QP30	POWER3	0	3500	1.17	0	1.17	0	1C4h	180h	4	4		100h	60	0	0	x
6	LA3.QP20	POWER3	0	3500	1.17	0	1.17	0	1C4h	180h	5	5		100h	60	0	0	x
7	CB0.QP20	POWER3	0	3500	1.17	0	1.17	0	1C4h	180h	6	6		100h	60	0	0	x
8	CB0.QP30	POWER3	0	3500	1.17	0	1.17	0	1C4h	180h	7	7		100h	60	0	0	x
9	CB0.QP40	POWER3	0	3500	1.17	0	1.17	0	1C4h	1A0h	8	8		100h	60	0	0	x
10	CC0.QP20	POWER3	0	3500	1.17	0	1.17	0	1C4h	1A0h	9	9		100h	60	0	0	x
11	CC0.QP30	POWER3	0	3500	1.17	0	1.17	0	1C4h	1A0h	10	10		100h	60	0	0	x
12	CC0.QP40	POWER3	0	3500	1.17	0	1.17	0	1C4h	1A0h	11	11		100h	60	0	0	x
13	CD0.QP20	POWER3	0	3500	1.17	0	1.17	0	1C4h	1A0h	12	12		100h	60	0	0	x
14	CD0.QP30	POWER3	0	3500	1.17	0	1.17	0	1C4h	1A0h	13	13		100h	60	0	0	x
15	CD0.QP40	POWER3	0	3500	1.17	0	1.17	0	1C4h	1A0h	14	14		100h	60	0	0	x
16	CD0.QP60	POWER3	0	3500	1.17	0	1.17	0	1C4h	1A0h	15	15		100h	60	0	0	x
1	LC0.QP20	POWER2	0	3500	1.17	0	1.17	0	2C4h	290h	16	16		100h	60	0	0	x
17	LA1.QP20	POWER3	0	3500	1.17	0	1.17	0	1C4h	190h	16	16		100h	60	0	0	x
18	LA1.QP30	POWER3	0	3500	1.17	0	1.17	0	1C4h	190h	17	17		100h	60	0	0	x
19	LA1.QP40	POWER3	0	3500	1.17	0	1.17	0	1C4h	190h	18	18		100h	60	0	0	x
20	LA1.QP50	POWER3	0	3500	1.17	0	1.17	0	1C4h	190h	19	19		100h	60	0	0	x
21	LA2.QP20	POWER3	0	3500	1.17	0	1.17	0	1C4h	190h	20	20		100h	60	0	0	x
22	LA2.QP30	POWER3	0	3500	1.17	0	1.17	0	1C4h	190h	21	21		100h	60	0	0	x
2	RC0.QP20	POWER2	0	3500	1.17	0	1.17	0	2C0h	260h	8	8		100h	60	0	0	x
3	RC0.QP30	POWER2	0	3500	1.17	0	1.17	0	2C0h	260h	9	9		100h	60	0	0	x
4	LA0.KI70	POWER2	0	6500	0.63	0	0.63	0	2C0h	260h	14	14		100h	60	0	0	x

EQPMOD.CSV

MOVE	0	0	255	Invalid	Invalid	0
MOVE	255	0	0	Missing		1
MOVE	0	255	0	Running		2
MOVE	0	0	255	Standby		4
MOVE	255	0	0	InterLock		8
MOVE	255	255	0	Local		16
MOVE	255	0	255	Fault		32
MOVE	255	0	0	End Left		64
MOVE	255	0	0	End Right		128
MOVE	255	0	0	Wire-grid	Grid	256
MOVE	255	0	0	Slit	Slit	512
MOVE	255	0	0	In Target		1024
MOVE	0	255	0	Out of Target		2048
MOVE	255	0	0	In	In	4096
MOVE	0	255	0	Out	Out	8192
MOVE	255	255	0	Half	Half	16384
MOVE	255	0	0	End Center		32768
MPOWP	0	0	255	Invalid		0
MPOWP	255	0	0	Off	Off	1
MPOWP	0	255	0	On	On	2
MPOWP	0	0	255	Standby	Standby	4
MPOWP	255	0	0	InterLock		8
MPOWP	255	255	0	Local		16
MPOWP	255	0	255	Fault		32
MPOWP	0	0	0		Reset	64
MSG	0	0	255	Illegal		0
MSG	255	0	0	In the Beam	In	1024
MSG	0	255	0	Out of Beam	Out	512
POWC	0	0	255	Invalid		0
POWC	255	0	0	Off	Off	1
POWC	0	255	0	On	On	2
POWC	0	0	255	Standby	Standby	4
POWC	255	0	0	InterLock		8
POWC	255	255	0	Local		16
POWC	255	0	255	Fault		32
POWC	0	0	0		Reset	64
POWC	255	0	0	In	In	512
POWC	0	255	0	Out	Out	1024
POWP	0	0	255	Invalid		0
POWP	255	0	0	Off	Off	1
POWP	0	255	0	On	On	2
POWP	0	0	255	Standby	Standby	4
POWP	255	0	0	InterLock		8
POWP	255	255	0	Local		16
POWP	255	0	255	Fault		32
POWP	0	0	0		Reset	64
POWSIM	0	0	255	Invalid		0
POWSIM	255	0	0	Off	Off	1
POWSIM	0	255	0	On	On	2
POWSIM	0	0	255	Standby	Standby	4
POWSIM	255	0	0	InterLock		8
POWSIM	255	255	0	Local		16

FecName	Protocol	FecNetwork	FecNode	IPAddr	IMHard	AlarmCheck	CamacArMaxCrate	Loop1	Loop2	Loop3	ADCaddr	MaxADC	IRQvec1
GPSPMAG	1	000000C0	00001B312EDC			1							
DENMARK	1	000000C0	44464900DBA9				280H	1 NAC1					
MCR	1	000000C0	08004E0181E7										
FECSIM	1	000000C0	0000C0A93414										
ITALY	1	000000C0	08004E0245E9										
GJFGJF	1	000000C0	00AA0000DD562										
INSIR1	1	000000C0	00001B3943B4								280H	1	12
POWER1	1	000000C0	08004E033A22								280H	1	12
NIGERIA	1	000000C0	08004E032CEE										
ITALIA	1	000000C0	00AA0000DD4E5								300H	1	12
SIMTAPE	1	000000C0	08004E034059								300H	1	12
DEFAULT	2			PMPS38	1								
MONITOR	1	000000C0	08004E02975F										
NORWAY	1	000000C0	08004E02560A										
PACE	1	000000D0	00001B1811E2										
EDWARD	1	000000C0	08004E0161F6										
POWER2	1	000000C0	08004E032A8E										
POWER3	1	000000C0	08004E033AD5										
KLAAS	1	000000C0	08004E03463F										
RUEDIGER	1	000000C0	08004E033840										

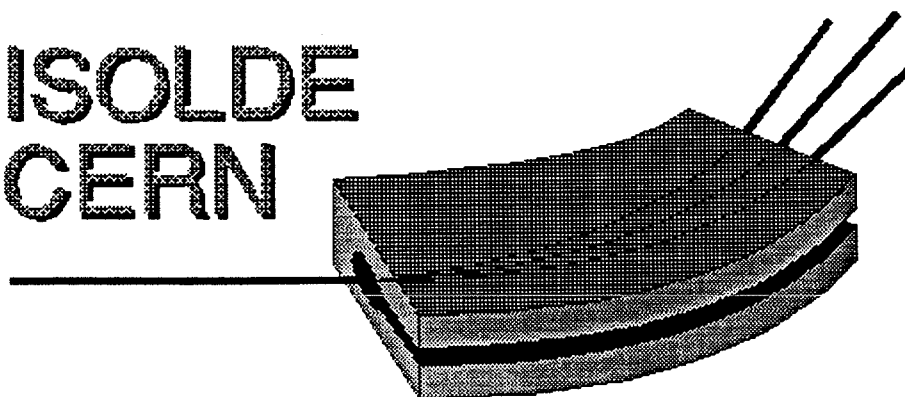
%	Isolde Equipment Database		Used by applications			Used by applications		
	Name	Type	FecName	EqpModu	EqpNumb	Unit	Increment	Control
	GPS.HT	60 KV Fine	NIGERIA	ASTECV	1	Volt		10 l:\ht\ht.exe
	GPS.HTMEAS	60 KV Measure	NIGERIA	MEASV	1	Volt		10 l:\ht\htmeas.exe
	GPS.TARGET	Ion Source Supply	DENMARK	POWC	5	Volt		0.1 HtSupply
	GPS.LINE	Ion Source Supply	DENMARK	POWC	7	Volt		0.1 HtSupply
	GPS.FILMNT	Ion Source Supply	DENMARK	POWC	9	Volt		10 HtSupply
	GPS.ANODE1	Ion Source Supply	DENMARK	POWC	2	Volt		10 HtSupply
	GPS.ANODE2	Ion Source Supply	DENMARK	POWC	3	Volt		10 HtSupply
	GPS.SRCMAG	Ion Source Supply	DENMARK	POWC	4	Volt		1 HtSupply
	GPS.OVEN	Ion Source Supply	DENMARK	POWC	6	Volt		0.1 HtSupply
	GPS.ELBOMB	Ion Source Supply	DENMARK	POWC	8	Volt		10 HtSupply
	GPS.ELDEFL	Ion Source Supply	DENMARK	POWC	1	Volt		10 HtSupply
	GPS.TEMP1	Thermocouple	DENMARK	POWC	10	Celsius		10 Temp
	GPS.TEMP2	Thermocouple	DENMARK	POWC	11	Celsius		10 Temp
%		GPS.TEMP3	Thermocouple	FECSIM	POWSIM		12 Celsius	10 Temp
%		GPS.COOL	Target Cooling	FECSIM	POWSIM		16 N/A	10 OnOff
%		GPS.TRAFO	Insul. Transforme	FECSIM	POWSIM		17 Amps	10 HtSupply

Appendix E : The ISOLDE Control System Reference.

The Isolde Control System Reference

A. Bret, I. Deloose, G.Leo, A. Pace and G. Shering

**ISOLDE
CERN**



Isotope Separator On Line

Aarhus, Argonne, Athens, Atlanta, Bergen, Berkeley, Berlin, Bielefeld, Bombay, Bonn, Brunswick, Caen Ganil, Caltech, Chalk River, Copenhagen, Darmstadt, Delft, Wuerenlingen, Erlangen - Nuernberg, Florence, Geneva, Gent State, Aalborg, Giessen, Gothenburg, Groningen, Harvard, Juelich, Karlsruhe, Kassel, Konstanz, Kyoto, Leuven, Lisbon, Lund, Lyon, Madrid, Mainz, Manchester, Maryland, McMaster, Montreal, Munich, Muenster, Nagoya, New York, Orsay, Oslo, Oxford, Paris, Princeton, Psi, Rossendorf, Rutgers, Sacavem, Saclay, Sheffield, Sofia, Strasbourg, Stockholm, Studsvik, Surrey, Tel-Aviv, Toronto, Troitzk, Uppsala, Valencia, Victoria, Warsaw, Zagreb, Zuerich and Cern.

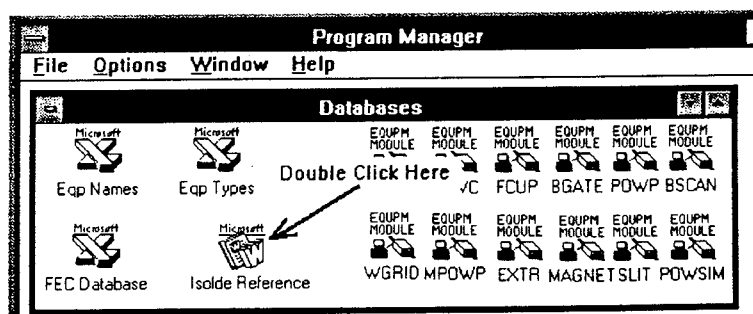
Table of contents

Introduction.....	III
1. The Isolde Disk Server	IV
2. Console Access to the Equipment.....	VI
Reading the Hardware: Hot and Cold Links	VI
Reading the Hardware from Excel	VIII
Reading one Value	VIII
Reading data using the Excel Macro Language	IX
Reading Arrays	X
Reading Arrays using the Excel Macro Language	XII
Reading the Hardware from Visual Basic	XII
Reading the Hardware from Microsoft C	XIII
Synchronous RPC (DDESyncRPC)	XIV
Asynchronous RPC (DDECreateRPCHotlink)	XV
Writing the hardware: Poke commands	XVII
Writing the Hardware with Excel.....	XVII
Writing one Value.....	XVII
Writing Array of Data	XVIII
Writing the Hardware with Visual Basic.....	XVIII
Writing the Hardware with Microsoft C.....	XIX
Creating and attaching Knobs: DDE Execute	XIX
Creating a Control Panel.....	XXI
Example - Attaching Knobs from Excel.....	XXIII
Example - Attaching Knobs with Visual Basic.....	XXIII
The RPC Server	XXIV
Alarms Program.....	XXIV
DDE Server	XXV
3. The Error Server.....	XXVI
4. Access to Excel Databases.....	XXVIII
Functions for reading the CSV databases	XXX
5. Writing Application Programs	XXXII
Using Windows Nodal	XXXII
Using Microsoft Excel	XXXIII
Using the Windows Software Development Kit	XXXIV
6. Equipment Module Definition	XXXV
The equipment module file	XXXV
Choosing the Equipment Module Properties.....	XXXVI
Digital Status Control: STCC and STAQ	XXXVI
Analog Status Control: CCV and AQN	XXXVII
ADCs and DACs Scaling Factors.....	XXXVIII
Properties Documentation.....	XXXVIII
Testing and debugging the Equipment module.....	XXXVIII
Equipment Module Initialization.....	XL
6. The Front End Computer	XLIII
8. Network Call translation.....	XLVI
9. Isolde Names.....	XLVIII
Annex: Equipment Module List and Element Database	L
Example of an Equipment Module listing	L

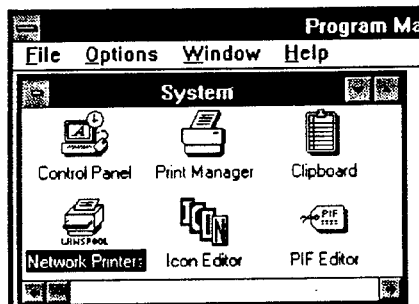
Introduction

This paper does not pretend to cover the Isolde Control System neither in part, neither in whole. Its purpose is just to introduce the users and all persons involved in its setup to its concepts.

Experience has shown that this document has a certain tendency to become obsolete rapidly. You can at any moment read the latest version by clicking on the *Isolde Reference* icon of the *Database* program Group of the Isolde Console on the Isolde Server. From the PS Network you can read this document by issuing a login `isolde/guest` command to the dos prompt.



This document can be easily printed on the printer of your choice that can be selected with the applications *Network Printers (Winspool)* and *Control Panel*.



1. The Isolde Disk Server

The Isolde Disk Server is a 80386-based personal computer with a 300 MBytes hard disk running Novell Netware. The partition used for the Isolde Control System is in the volume `isolde/usr:` and is 255 Mbytes big. The remaining disk space is in volume `isolde/sys:` and is used only by the server system. The volume `isolde/usr:` is mapped to several DOS drives when a workstation is attached. These are:

Drive L:

Contains all source and executable files of the control system. Files are organized in subdirectories. For example

- L:\FECNOD contains nodal sources
- L:\RPC contains the Remote Procedure Call (RPC) library
- L:\DATABASE contains the Database access library
- L:\DDESRV contains the bridge between Dynamic Data Exchange (DDE) and Remote Procedure Call (RPC)
- L:\EQPMOD contains all the Equipment Module Sources

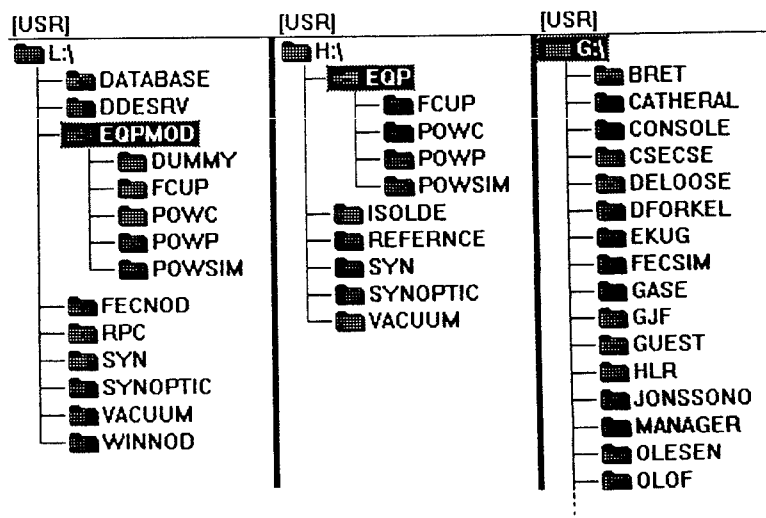
Drive H:

Contains all database files, either in Excel Format or Comma Separated Value (CSV). Both formats are Excel compatible. Files are organized in subdirectories. For example

- H:\EQP Equipment Database (Names, Front End Table etc ...)
- H:\EQP\xxxx Equipment Module database (Table and Specification)
- H:\REFERENCE This manual

Drive G:

The home drive. Every registered user has a subdirectory for development. Every registered FEC has a subdirectory where the FEC-specific version of nodal is stored.



This directory tree can be easily seen and explored using the *File Manager* utility under Windows.

2. Console Access to the Equipment

Any Windows application supporting Dynamic Data Exchange (DDE), as well as any program written in C, Visual Basic or Nodal can directly access any Isolde Equipment. The only requirement is to have the `RPCSRV.EXE` and the `ERRORSRV.EXE` programs running.



RPC Server



Error Server

In the following sections it will be explained how to read and write from and to the hardware and to attach Knobs. For every function some examples using Microsoft Excel, Visual Basic and C will be provided.

Reading the Hardware: Hot and Cold Links

An *hot-link* is a permanent connection between an application cell or a variable of a program and an equipment property. The value is updated whenever the value of the property is changed. Once the hot link is established, the `RPCSRV` program will issue an equipment call periodically (from 500 ms to several minutes period) and refresh the application cell or the variable when the value is changed.

A *cold-link* is a single call connection. The value is updated only once, when you ask the cold-link. After it, the connection is closed.

The reading of the hardware can be executed establishing both hot or cold connections. The way the link is established is application dependent and is described in the client application manual. The data exported from `RPCSRV` are all under the DDE topic *DdeSrv/Main* (*DdeSrv* is the application name while *Main* is the topic) and requests have the following formats:

EquipmentName_Property_RefreshPeriod_SizeSpecification_Format

EquipmentName and *Property* are mandatory for any acquisition, while *RefreshPeriod*, *SizeSpecification* and *Format* can be omitted to be equivalent to their default values. For example `GPS.QP030_AQN` is a valid request to obtain the acquired voltage of the `GPS.QP030` lens. The voltage will be returned as a ASCII string containing the value and, if an hot link is specified, will be refreshed every 5000 milliseconds, the default refresh period. The default refresh period can be modified in the `RPCSRV`.

RefreshPeriod is the period in milliseconds for the desired refreshment rate. For example `GPS.QP030_AQN_1000` will force a refreshment rate of 1 second of the data in the client application. The value of 0 can be used to force a cold link instead of an hot link: `GPS.QP030_AQN_0` means that the voltage of the `GPS.QP030` lens is acquired only once and will never be refreshed.

SizeSpecification is used to acquire arrays of data. The field has one of the four following forms: 'HpVn', 'VnHp', 'Vn' or 'Hp' where n represents the number of rows of the array and p its number of columns. For example, the following array formula:

CA0.SC20_READSCAN_H3V5

request $3 \times 5 = 15$ points of the last trace of the beam scanner. The values will be provided in a character string and can be seen as a 3×5 table: <Tab> between values are used as columns separators, while <CR> are used to separate columns.

The data for CA0.SC20_READSCAN_H2V3 (6 value) will come as:

value	<tab>	value	<tab>	value	<cr>	value	<tab>	value	<tab>	value
-------	-------	-------	-------	-------	------	-------	-------	-------	-------	-------

The data is then refreshed at the specified period or at the default period is unspecified: Both CA0.SC20_READSCAN_H3V5_10000 or CA0.SC20_READSCAN_10000_H3V5 will refresh the array every 10000 milliseconds.

If unspecified, N and P are supposed to be equal to 1. Therefore, row and column properties are able to be acquired by specifying just one parameter. For example, in the following formulas:

GPS.QP030_AQN_10000_h3
GPS.QP030_AQN_10000_v5

The first one acquires 3 elements and returns them in a row (<TAB> separated), while the second one acquires 5 elements in a column (<CR> separated).

GPS.QP030_AQN is of course equivalent to GPS.QP030_AQN_H1V1.

The last field, *Format*, is used to specify the way the data is represented in the string of data sent from the RPCSRV to the client application. The field has the form 'Fx', where x is the format requested. Note that the Windows Clipboard format is always text (CF_TEXT).

FS is the default value of the format. GPS.QP030_AQN is equivalent to GPS.QP030_AQN_FS and a value of 14.2 is returned in a string as "14.2". GPS.QP030_AQN_H2 is equivalent to GPS.QP030_AQN_H2_FS and two values of 94.2,97.3 are returned in a string as "94.2\t97.3", ascii codes: 57,52,46,50,9,57,55,46,51.

The format FC is used to send every value in one character only. The range is limited in (0,255) or (-127,128). With GPS.QP030_AQN_FC_H2, two values of 94.2,97.3 are returned in a string as ascii codes: 94,97 or "^a". Note that also 606.34,-159.0 will also be returned as 94,97.

The format FD is used to send every value in two bytes only. the range is limited in (0,65535) or (-32767,32768). With GPS.QP030_AQN_FC_V2, two values of 10094.2,40097.3 are returned in a string as ascii codes:

39,110,156,161 ($39*256+110=10096$, $156*256+161=40097$) or "'n£1". Note that overflow errors are also possible as in the case of single bytes values.

The last format, FF, is used to send every value in 8 bytes double precision values. With GPS.QP030_AQN_FF_V2, two values of 10094.2,40097.3 are returned in a string "xxxxxxxxxxxxxxxx", where xxxxxxxx and yyyyyyyy are the representation of 10094.2 and 40097.3 in 8 bytes double precision format.

All options in the DDE requests that are optional can be mixed in order:
 GPS.QP030_AQN_3000_FF_H3V2 is equivalent to
 GPS.QP030_AQN_FF_V2H3_3000.

In the next section some examples using Microsoft Excel, Microsoft Visual Basic and Microsoft C will be provided.

Reading the Hardware from Excel

Reading one Value

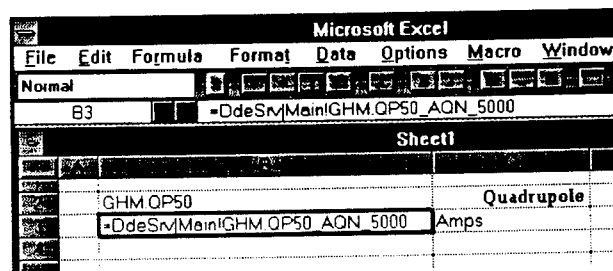
To read one Element property in Excel, the formula to be typed into a cell is:

=DDESRV|MAIN!EquipmentName_Property_Timer

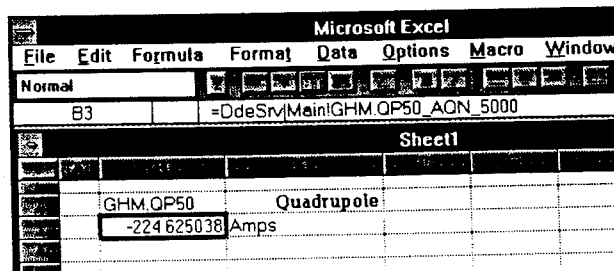
For example:

=DDESRV|MAIN|GPS.QP030_AQN_1000

will attach the acquisition (AQN) of the quadrupole in the current cell and this acquisition will be refreshed every 1000 milliseconds, see next figure:



After a while



If the Timer field has not been specified by user, the acquisition will happen every Default Refresh Period. The Default Refresh Period may be changed by

selecting the *Acquisition and Refresh Periods* command in the *Options* menu in the DDE Server window. The *Timer* value should be of course a strictly positive integer, since it represents a time interval.

The default *Timer* is equal to 5000 milliseconds.

It is allowed to fix the *Timer* field to '0'. In this case a cold-link will be established: the acquisition will be done just once. This may be useful when some values of an equipment has to be acquired punctually.

For example, the following formula :

```
= DdeSrv|Main!GPS.TEMP1_AQN_0
```

will display *just once* the temperature associated to the thermocouple GPS.TEMP1.

If the *Timer* field value is lower than the Minimum Refresh Period, the acquisition will be performed every Minimum Refresh Period. This parameter is also configurable in the *Options* menu of the DDE Server.

Reading data using the Excel Macro Language

An other way to acquire data from an equipment, given a property, is to write a simple macro like the following one:

```
Send_RPC_Read_To_Hardware
Channel=INITIATE("DdeSrv", "Main")
Value=REQUEST(Channel, "EquipmentName_PropertyName_0")
=TERMINATE(Channel)
=RETURN()
```

The *INITIATE* function opens a logical channel between Excel and an other Windows applications. Its first argument specifies the name of the application to be accessed, "DdeSrv". The second one specifies the name of the topic to be accessed within the remote application. It has to be "Main".

The purpose of the *REQUEST* function is to retrieve the value of an item within the application connected to the channel Channel. In the case of the DDE Server, the item consists of the couple (EquipmentName, PropertyName). The option '_0' is compulsory because it means that the data is asked just once. The target value appears then in the cell where the *REQUEST* instruction has been typed.

For example, the instruction :

```
= REQUEST (Channel, "GPS.VS10_AQN_0")
```

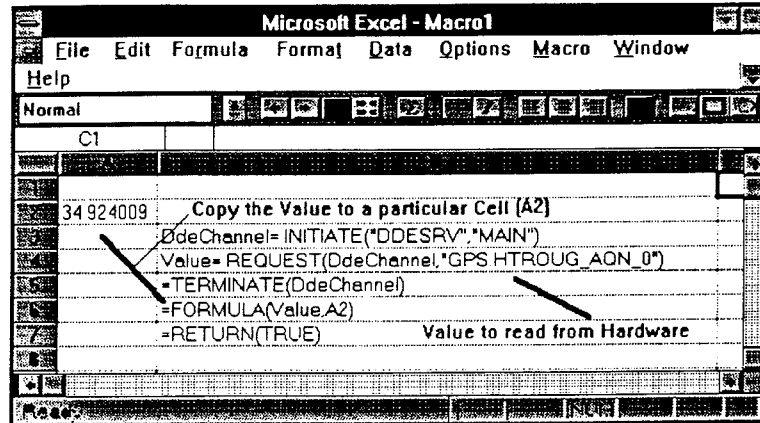
will retrieve the pressure in the vacuum section number 10, while

```
= REQUEST (Channel, "GPS.VS10_STAQ_0")
```

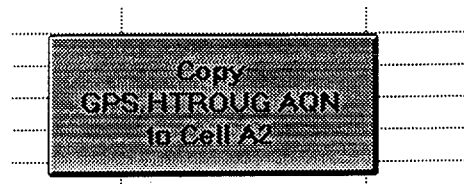
will retrieve its status (On, Off, Fault).

The *TERMINATE* function closes the logical link established between Excel and the remote application (DdeSrv). Every channel must be closed if it is no longer used any more because the number of channels open simultaneously to/from Excel is limited.

The next figure shows an example of a simple macro:



The Macro can be attached to a particular Button:



The difference between using macros or hot links to read the hardware is that with the macros the hardware is read once, while, with the hot-link, the acquisition is generally periodic.

Reading Arrays

Arrays can also be easily linked. To do so, you have only to specify the number of elements you want from the RPC call and in which format (row or column) you want to receive them.

To hot-link an array, the formula syntax is the following one :

```
=DdeSrv|Main!EquipmentName_PropertyName_Timer_DataDisplay
```

The *DataDisplay* field must take one of the four following forms : 'hPvN', 'vNhP', 'vN' or 'hP' where N represents the number of rows of the array and P its number of columns.

A valid request is for instance the following array formula:

```
= DdeSrv|Main!GPS.QP030_AQN_10000_h3v5
```

This will attach the array acquisition (AQN) of the quadrupole to an Excel 3 x 5 table and this table will be refreshed every 10000 milliseconds.

If unspecified, N and P are supposed to be equal to 1. Therefore, row and column properties are able to be acquired by specifying just one parameter. For example, in the following formulas:

```
= DdeSrv|Main|GPS.QP030_AQN_10000_h3
= DdeSrv|Main|GPS.QP030_AQN_10000_v5
```

The first one acquires 3 elements and returns them in a row, while the second one acquires 5 elements in a column.

For example:

The formula `=ddesrv|main|GPS.QP010_AQN_5000_H10` will read an array of 10 acquisitions from the equipment module of GPS.QP10. The Equipment module should support array calls, otherwise an *illegal read write attempted* error is returned. The 10 acquisitions are returned in a row (H10 was specified). The formula `=ddesrv|main|GPS.QP010_AQN_5000_V10` would do the same but return the values in a column. Finally, writing `=ddesrv|main|GPS.QP010_AQN_5000_H5V2` is equivalent to read 10 elements and return them in a 5 x 2 table.

Microsoft Excel - Sheet2						
File Edit Formula Format Data Options Macro Window						
Normal						
B18 X ✓ =ddesrv main GPS.HTFINE_AQN2_5000_H4						
A	B	C	D	E	F	G
17						
18	5030					
19						
20						

To do so:

- Select a Cell range, like B18:E18
- type in the formula as described in the previous paragraph. In this case, the result must be sent in a row of 4 elements (it ends with _H4)
- press <Ctrl>+<Shift>+<Enter> (see Excel Manual)

Microsoft Excel - Sheet2						
File Edit Formula Format Data Options Macro Window						
Normal						
B18 {=ddesrv main GPS.HTFINE_AQN2_5000_H4}						
A	B	C	D	E	F	G
17						
18	422.186	423.445	435.435	418.567		
19						

The array is transferred (once every 5 seconds, as requested).

Normal			
B22 X ✓ =ddesrv main GPS.HTFINE_AQN2_5000_V4			
A	B	C	D
21			
22			
23			
24			
25			

Normal		
B22 {=ddesrv main GPS.HTFINE_AQN2_5000_V4}		
A	B	C
21		
22	429.119	
23	455.931	
24	414.608	
25	441.434	

Reading Arrays using the Excel Macro Language

To read arrays or tables of data using the Excel Macro Language the syntax is identical to the one used for the transfer of single values. The *REQUEST* macro-function needs to be written the following way:

```
=REQUEST(Channel,"EquipmentName_PropertyName_0_DataDisplay")
```

For instance, the instruction :

```
= REQUEST (Channel, "GPS.VS10_AQN_0_h5v3")
```

will retrieve a 5 x 3 table of pressures in the vacuum section number 10.

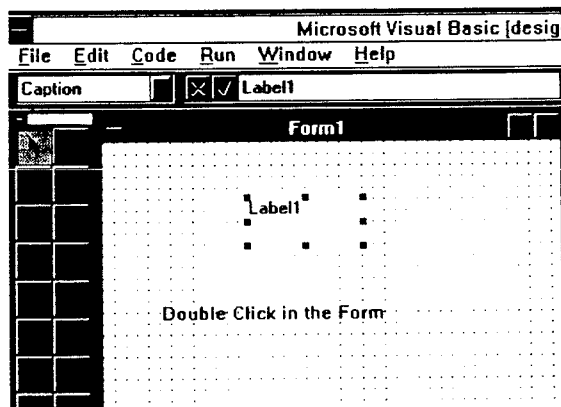
Reading the Hardware from Visual Basic

If you want your Visual Basic applications to use data supplied by the Hardware, then you can create a *client link*. A client link can be associated to a text box, label, or picture box. In the code relative to one of the commands mentioned you must declare, before starting the link, the link topic, the link item and the link mode.

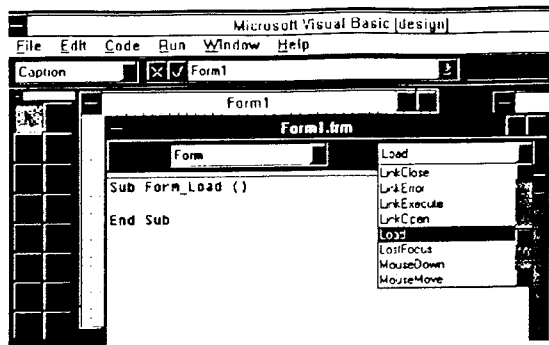
LinkTopic, LinkItem and LinkMode are properties of the command. The link starts when LinkMode is different than 0, and that property also define the kind of link. If LinkMode is set to 1 then a hot-link is started, else if it is set to 2 a cold link is executed. If you setup a cold link the value is refreshed every time you ask a *Request*. It is recommended to check that the LinkMode property is set to 0 before changing LinkTopic or LinkItem properties.

For example:

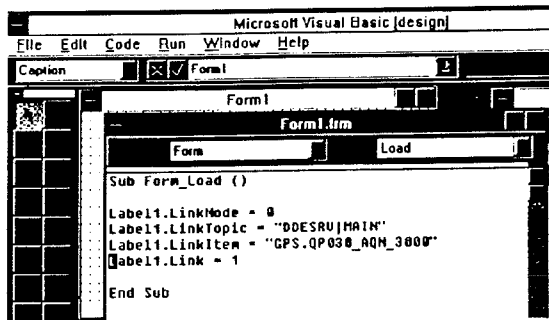
If you want a Label to show the AQN value of the quadrupole GPS.QP030, and this value to be refreshed every 3 seconds, you simply doubleclick inside the form:



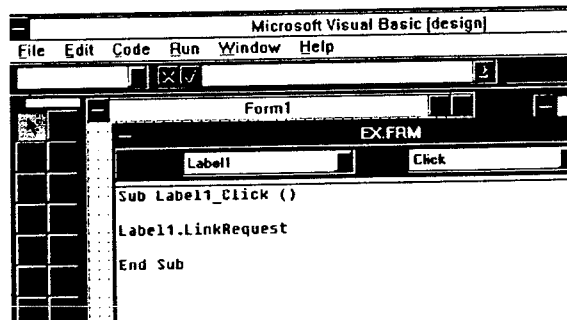
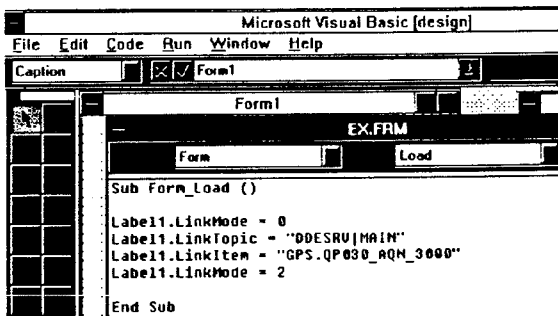
Then, you should select the event that start the hot link. As long as you want that value permanently displayed the correct event is the Load one:



Finally you can write the software to set the hot link. In this example we begin setting to 0 the LinkMode property. The next step consists in the declaration of LinkTopic and LinkItem, in DDE format. After that we can start the hot-link:



If otherwise you want to refresh a value every time you click in the command you should set a cold link and add some code at the Click event of the command Label1:



Reading the Hardware from Microsoft C

An other way of accessing the equipments from Windows is to use the C language and the DDE functions available in the L:\DDEFUNCS\DDEFUNCS.LIB library. The program should also include

```
#include "l:\ddefuncs\ddefuncs.h"
```

This file contains the following function prototypes and constants.

Synchronous RPC (DDESyncRPC)

```
int
DDESyncRPC (ElemName, Property, ElemSize, DataPointer, Datasize, DataFormat);

char *ElemName;
char *Property;
int ElemSize;
void far *DataPointer;
int DataSize;
int DataFormat;
```

A synchronous RPC executes the RPC immediately and waits for the return values from the FEC. The function returns the completion code and the valid data in DataPointer.

The use *on entry* is :

ElemName - the pointer to a string containing the name of the element.

Property - the pointer to a string containing the property.

ElemSize - the number of values to read (positive) or write (negative).

DataPointer - a pointer to the allocated memory.

DataSize - size in bytes of the allocated memory.

DataFormat - the format of the data in DataPointer. Actually the following formats are implemented :

- CF_TEXT : text format. Data is separated by tabs.
- CF_DOUBLE : the data is read or written in the 8 byte double format.
- CF_INTEGER : the data is read or written in the 2 byte integer format.
- CF_BYTE : the data is read or written in the 1 byte char format.

The use *on exit* is :

DataPointer - the allocated memory contains the data in case of a read RPC. If the format is CF_TEXT and the property is STCC or STAQ, it gives the real status name.

The return value is the completion code. It is zero when no errors occurred, otherwise it returns errors from the equipment (Hardware problems or Out of Range data) and from the RPC Server (Communication errors, Non existent Name).

For example :

```
#include "windows.h"
#include "l:/ddefuncs/ddefuncs.h"
#include "stdio.h"

WinMain(hInstance, hPrevInstance, lpCmdLine, nCmdShow)
HANDLE hInstance;
HANDLE hPrevInstance;
LPSTR lpCmdLine;
int nCmdShow;
{
    int code;
    double far value;
    char svalue[100];

    code=DDESyncRPC("BL.QP240", "AQN", 1, (void far *)&value, sizeof(double), CF_DOUBLE);
    if (code)
    {
        RegisterError("MyProgram", code, "BL.QP240", TRUE);
        return;
    }
    else printf("The current in BL.QP240 is %f\n", value);

    code=DDESyncRPC("BL.QP240", "STAQ", 1, (void far *)&svalue, sizeof(double), CF_TEXT);
    if (code)
    {
        RegisterError("MyProgram", code, "BL.QP240", TRUE);
        return;
    }
    else printf("The status of BL.QP240 is %s\n", svalue);
}
```

Asynchronous RPC (DDECreateRPCHotlink)

```

int
DDECreateRPCHotlink (Name, Prop, Timer, ElemSize, Format, Identifier, Request);

char *Name;
char *Prop;
long Timer;
int ElemSize;
int Format;
int Identifier;
BOOL Request;

```

A asynchronous RPC is used in application programs that want a continuous update of his elements. A DDECreateRPCHotlink is in fact a request to add the ElemName with the specified Property to an internal hot-link table. It means that the value will be read from the equipment and sent to your application with an interval specified by the Timer. An example is the Synoptic program (e.g. Booster Transfer Line). The client (application program) doesn't need a timer, since the data is posted to the window by a WM_DDE_DATA message captured by the DDEWindowProc function.

The use *on entry* is :

ElemName - the pointer to a string containing the name of the element.

Property - the pointer to a string containing the property.

ElemSize - the number of values to read (always positive).

Timer - interval in milliseconds between two equipments accesses.

DataFormat - the format of the data in DataPointer. Actually the following formats are implemented :

- CF_TEXT : text format. Data is separated by tabs.
 - CF_DOUBLE : the data is read or written in the 8 byte double format.
 - CF_INTEGER : the data is read or written in the 2 byte integer format.
 - CF_BYTE : the data is read or written in the 1 byte char format.
- Identifier - an own specified integer value, used as reference, when receiving the data, to update the correct item in your program.
- Request - True if you want to send a request message to the server in order to do an synchronous acquisition at the same time. Normally, it is not used in an C-program.

The function returns zero if your request is accepted.

The received data is handled by the DDEWindowProc function.

```

int DDEWindowProc (hWnd, message, wParam, lParam);

HWND hWnd;
WORD message;
WORD wParam;
LONG lParam;

```

On entry :

The 4 parameters are the values from the actual Window procedure.

On exit :

if the return value is positive then it is the identifier of the corresponding hot link. Now you can read the data using the DDEReadData function.

If negative then return from the Window function.
If Zero, just continue;

```
int DDEReadData(lParam,DataPointer,DataSize);
```

```
LONG lParam;  
void far *DataPointer;  
int DataSize;
```

On entry :

lParam - parameter taken from the Window procedure.
DataPointer - a pointer to the allocated memory.
DataSize - size in bytes of the allocated memory.

The return value is the completion code.

```
void DDERemoveRPCHotlink(Identifier);
```

```
int Identifier;
```

Identifier - value specified when DDECreateHotlink was called. If the identifier is -1, all the RPCs according to this application are removed.

For example :

```
#include "windows.h"  
#include "l:/ddefuncs/ddefuncs.h"  
#include "stdio.h"  
  
#define SIZE      2  
  
double Value;  
char Names[]={"GPS.TARGET","GPS.HTROUG"};  
  
WinMain(hInstance,hPrevInstance,lpCmdLine,nCmdShow)  
HANDLE hInstance;  
HANDLE hPrevInstance;  
LPSTR lpCmdLine;  
int nCmdShow;  
{  
    int code,ident;  
    long timer;  
    HWND hWnd;  
  
    hWnd = CreateWindow("Class","Window Title",  
        WS_OVERLAPPEDWINDOW,  
        CW_USEDEFAULT,CW_USEDEFAULT,  
        CW_USEDEFAULT,CW_USEDEFAULT,  
        NULL,NULL,hInstance,NULL);  
    if (!hWnd) return (FALSE);  
    ShowWindow(hWnd,nCmdShow);  
  
    timer=5000;  
    for (ident=0;ident<SIZE;ident++)  
    {  
        code=DDECreateRPCHotlink(Names[ident],"AQN",timer,1,CF_DOUBLE,ident,FALSE);  
        if (code)  
        {  
            RegisterError("MyProgram",code,Names[ident],TRUE);  
            return;  
        }  
    }  
}  
  
long FAR PASCAL MainWndProc(hWnd, message, wParam, lParam)  
HWND hWnd;  
unsigned message;  
WORD wParam;  
LONG lParam;  
{  
    char Buf[100];  
    int id;  
  
    if (id=DDEWindowProc(hWnd,message,wParam,lParam),id<0) return NULL;  
    if (id>0)  
    {  
        DDEReadData(lParam,(void far *)&Value,sizeof(double));  
        sprintf(Buf,"The current in %s is %lf",Names[id],Value);  
        return NULL;  
    }  
  
    switch (message)  
    {  
        case WM_DESTROY:  
            DDERemoveRPCHotlink(-1);  
            break;  
    }  
}
```

Writing the hardware: Poke commands

The POKE DDE command can be used to write to the hardware single values or array of values. The way the DDE Poke is executed is application dependent and is described in the client application manual.

The DDE Poke command should use the DDE topic *DdeSrv/Main* (*DdeSrv* is the application name while *Main* is the topic) and requests have the following formats:

EquipmentName_Property_RefreshPeriod_SizeSpecification_Format

EquipmentName and *Property* are mandatory for any acquisition, while *RefreshPeriod*, *SizeSpecification* and *Format* can be omitted to be equivalent to their default values. In particular *RefreshPeriod* is meaning less and should be omitted. The only supported format is FS that is the default: *Format* is also meaningless and should also be omitted.

In this way, the client application can poke to the RPCSRV the string "128.47" to GPS.QP030_CCV to write 128.47 to the CCV of the GPS.QP030 lens or can poke to the RPCSRV the string "128.47\t27.1" to XXX.YYYNNN_CCV_H2 to write a two value array {128.47, 27.1} to the CCV of the XXX.YYYNNN device.

For STCC properties, strong as "ON", "OFF", "OPEN", "CLOSE", ... etc. can be poked directly. The equipment will receive the translated value according to the H:\EQP\STCODES.CSV database.

Writing the Hardware with Excel

Writing one Value

The only means to send data from Excel is to write its owns macros. This can be done by following this model :

```
Send_RPC_Write_To_Hardware
Channel=INITIATE("DdeSrv","Main")
=POKE(Channel,"EquipmentName_PropertyName",Data_ref)
=TERMINATE(Channel)
=RETURN()
```

The purpose of the *POKE* function is to send data to the application connected to the channel 'Channel'. In the case of the DDE Server, the item consists of the couple {EquipmentName, PropertyName}.

For example, the instruction :

```
= POKE (Channel, "GPS.QP030_CCV", B1)
```

will send the value from cell B1 of the Excel sheet to the CCV of the quadrupole GPS.QP030.

The *POKE* function returns *TRUE* if successful or an error status if the remote command has failed.

Writing Array of Data

Sending an array is possible by calling the *POKE* function just once :

```
Send_RPC_Write_To_Hardware
Channel=INITIATE("DdeSrv","Main")
=POKE(Channel,"EquipmentName_PropertyName_DataDisplay",Data_ref)
=TERMINATE(Channel)
=RETURN()
```

The *DataDisplay* field must take one of the four following forms : 'hPvN', 'vNhP', 'vN' or 'hP' where N represents the number of rows of the array and P its number of columns. If unspecified, N and P are supposed to be equal to 1. Therefore, row and column properties are able to be sent to the hardware by specifying just one parameter.

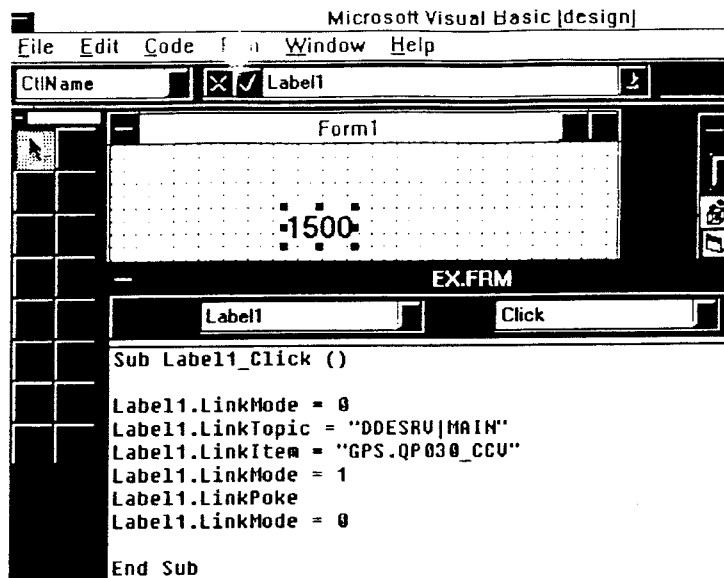
For example, the instruction :

```
= POKE (Channel, "GPS.QP030_CCV_h2v3", B1:C3)
```

will send a 2 x 3 array from the range of cells B1:C3 to the CCV of the quadrupole number 30.

Writing the Hardware with Visual Basic

Although the flow of data in a DDE conversation is usually from the server application to the client application, the client can also send data to the server. This is called poking data, and it is done with the LinkPoke method. LinkPoke sends the contents of the control to the server application. In the example below the value in the label is sent to the hardware each time we click the command.



Writing the Hardware with Microsoft C

For what regards writing the hardware in C you can refer to the section *Reading the Hardware with Microsoft C* because the syntax is the same. The parameter that specifies that you want to write a value is the ElemSize. In the case you want to write to the hardware, the number of values to write should be negative.

Creating and attaching Knobs: DDE Execute

Knobs are control panels that are serviced by the RPC Server. You can create new knobs or attach knobs to your applications. After a brief description of the knobs you can use we will see how to create new knobs and how to attach them to your applications.

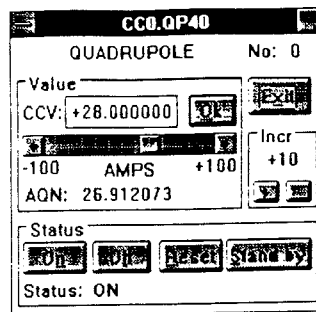
To attach a Knob you should use the following DDE Execute commands:

ElementName_ATTACH	- to attach a object-dependent knob to a particular object
ElementName_ICONIC	- to minimize an opened control panel
ElementName_RESTORE	- to restore a minimized control panel

The type of Control Panel that is opened when this command is received depends on the object and, to be more precise, it is read from the contents of the "Control" field of the H:\EQP\EQPNames.CSV database.

In particular, this value can be:

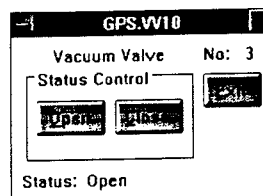
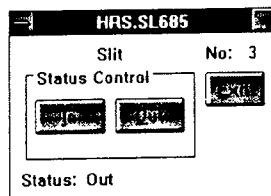
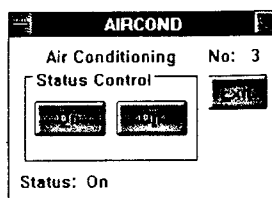
"Analog", for an analog control Panel



"OnOff", for an On/Off control Panel

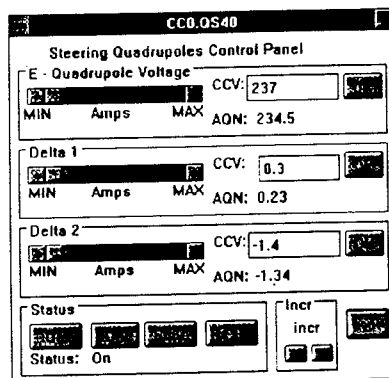
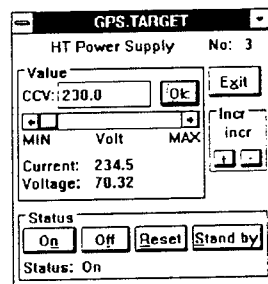
"InOut", for an In/Out control Panel

"OpenClose", for an Open/Close control Panel

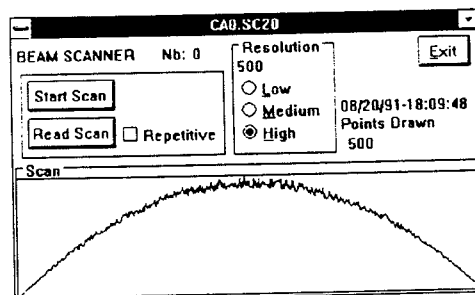


"HTSupply", for an analog control Panel for double acquisition (used for the high tension power supplies).

"SteerQuad", for an Control Panel designed for the Steering Quadrupoles



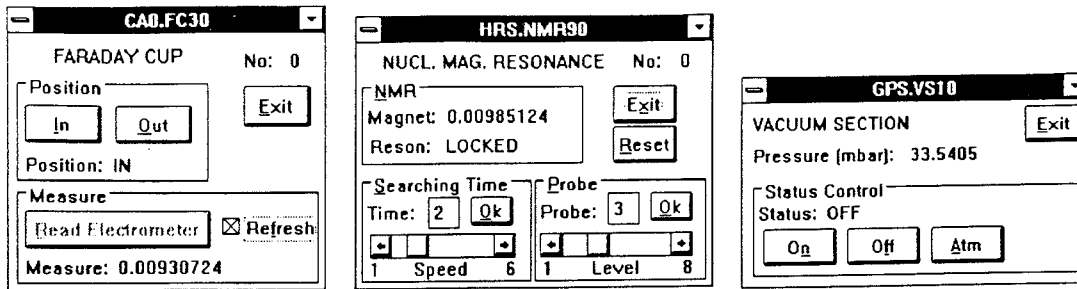
"Scanner", for an Control Panel designed for the Beam Scanners



"Fcup", for an Control Panel designed for the Faraday Cups

"NMR", for an Control Panel designed for the Nuclear Magnetic Resonance measurement

"Vacuum", for an Control Panel designed for the Vacuum Sections



Several other control panel are available and those shown here are just examples.

Creating a Control Panel

It's possible to create your own defined panel, compile and link it together with the main RPCSRV modules.

1/ Creating a local copy of the RPC Server.

Put the DOS prompt in the directory where to add the RPCSRV directory. Then call a batch named CTRPCDIR (CReateRPCDIRectory). This will create the RPCSRV directory, the two subdirectories SOURCE and KNOBS; and copies all the source files from the installed version L:\RPCSRV. Afterwards it will automatically generate all the necessarily files to be able to build the executable (see point 3).

2/ Creating a new knob and adding it to the RPC Server.

The subdirectory KNOBS contains all the control panels. Each panel consists of three files. The C-program (*.C), the dialog source (*.DLG) and the dialog resource (*.RES) file.

If you want to create a new knob, e.g. named DlgName, execute 'CTDIALOG DlgName'. This program will create the 3 standard files. It will create a standard dialog box with the description and sequence number fields, and also the Exit button. Use the Dialog Editor in Windows to edit the dialog box (using the KNOB-IDS.H include file).

The C-file contains three functions, starting with 'DlgName'.

a/ the Dialog message function. Called by Windows (DlgNameDialogFunc).

b/ the Initfunction. Called when the knob is created (DlgNameInit). Here you add all the asynchronous RPCs. To establish a continuous RPC that simultaneously updates a well defined control of your knob, use the AddRefreshTable function (see later).

c/ the Refreshfunction (DlgNameRefresh). This is optional and is called after the RPC server updated the elements of that control panel. This function allows you to add some non-standard refresh functions.

The second line is a special comment used to automatically generate the make files. The first element stands for the name of the functions DlgNameInit,

DlgNameRefresh. The second one is the string that corresponds with the EqpNames database Control field (capital letters). The third is the name of the dialogfunction (DlgNameDialogFunc).

3/ Generating the make files.

After you added a new knob or you changed the name of a function in a C-file, the make files has to be generated. Use 'MKRPCSRV -g' to do this. It will generate files for the Windows-SDK and the QCWIN compilers.

4a/ Building the RPC Server with the Windows-SDK compiler/linker.

Use 'MKRPCSRV' without option. The executable file RPCSRV.EXE is stored in the SOURCE subdirectory. Now, you can use CodeView (double screen) for debugging.

4b/ Building the RPC Server with the QCWIN compiler/linker/debugger.

Start QCWIN in Windows and open the RPCSRV.MAK project.

Functions available to establish and remove RPC hot-links, to be used in the dialog C-files.

```
int UpdateItem(hDlg,CtlID,DataPtr,Property);
```

```
HWND hDlg;
int CtlID;
char *DataPtr;
char *Property;
```

This function writes the data in the equipment and updates the control with that value.

Use on *entry* :

hDlg - window handle of the dialog box.

CtlID - control that has to be updated.

DataPtr - pointer to the memory that contains the data in string format (RPC_TEXT).

Property - property to be updated.

Return zero if no errors occurred.

```
int AddRefreshTable(hDlg,CtlID,Property,CallRefreshFunc);
```

```
HWND hDlg;
int CtlID;
char *Property;
BOOL CallRefreshFunc;
```

This function add a RPC hotlink and updates the control of a dialog box.

The use *on entry* is :

hDlg - window handle of the dialog box.

CtlID - identifier of the specified control (e.g. ID_CTL_DESC).

Property - property to read.

CallRefreshFunc - TRUE if the DlgNameRefresh function has to be called after the control has been updated.

It returns zero when no errors occurred.

```
void RemoveRefreshTable (hDlg,CtlID);
```

```
HWND hDlg;  
int CtlID;
```

This function removes a RPC linked with a control.

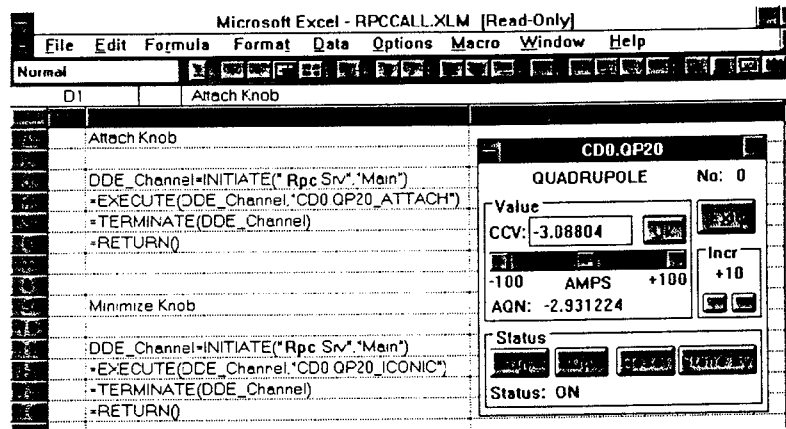
The use *on entry* is :

hDlg - window handle of the dialog box.

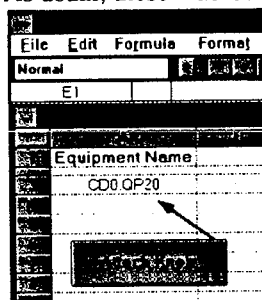
CtlID - identifier of the control which to remove the RPC. If -1 all the RPCs for that dialog box (hDlg) are removed.

Example - Attaching Knobs from Excel

The first macro in the next picture attaches a Knob to the Specified Element CD0.QP20, while the second one minimizes it.

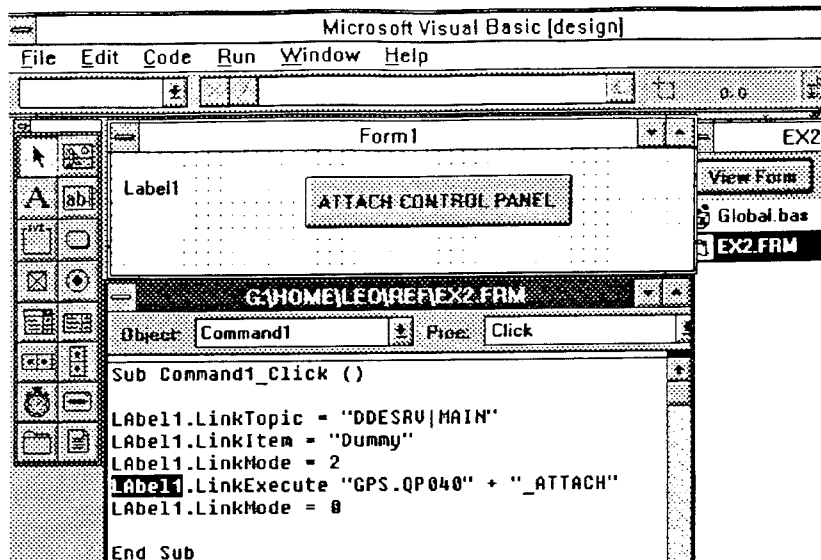


As usual, these macros can be attached to buttons



Example - Attaching Knobs with Visual Basic

In this example you can attach a control panel to the application by clicking the control ATTACH CONTROL PANEL.



The RPC Server

The RPC Server, alias RPCSRV, is a program that replaces the DDE Server, the Knobserver and the Alarms program. This program has been written in order to increase the equipment access speed & flexibility.

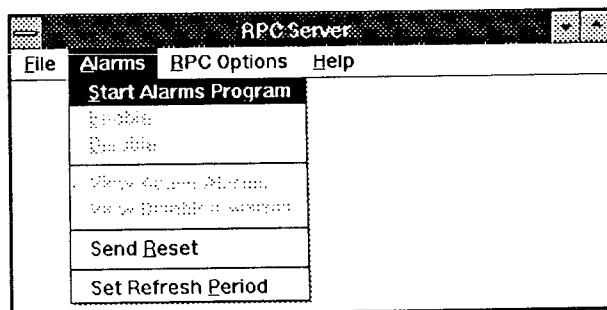


RPC Server

Alarms Program

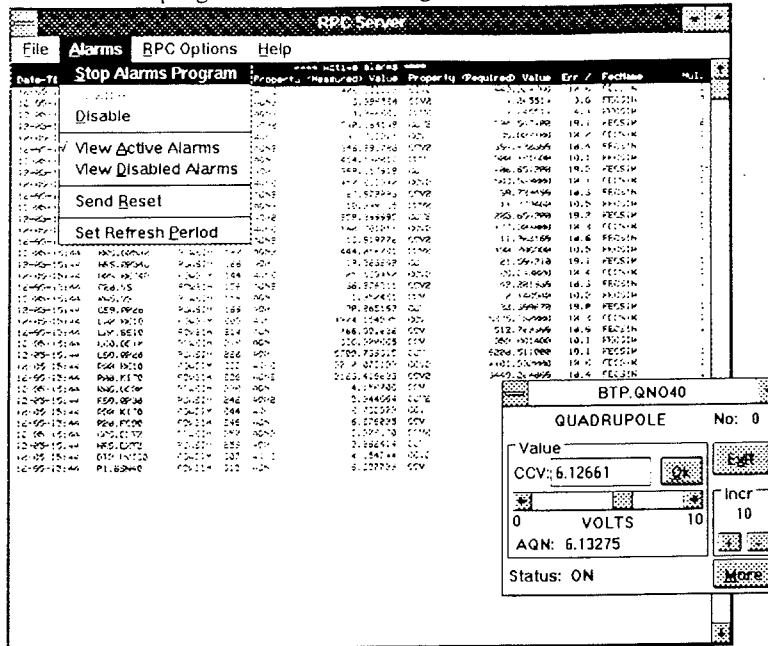
Select the Start Alarms Program option to start reading the alarms on the selected Front End Computers. To select a FEC, you have to set the AlarmCheck field in the FECADDR.CSV database to 1. If a FEC is not responding, the alarms program removes that computer from his list after one try.

The refresh period can be set in the Set Refresh Period dialog box (minimum value is 10000ms).



After the alarms program started, the window size changes to a fix setting to be able to display the alarms. Also the alarms menu changes. You select an alarmline by a single mouse-click (line inverted) in order to disable it or to send a reset (sends the required property). To display the disabled alarms, choose the View Disabled Alarms option.

A double mouse click will launch the control panel of that element.
At any time the alarms program can be stopped and restarted. After a restart, the program restores the original FEC-AlarmCheck list.



The alarms program reads only an alarm table from the FEC. On his part, the FEC compares the required and the measured properties of every initialized equipment; and fills up that table. The properties to be compared are defined in the COMPARE.CSV database.

Normally the program runs as an icon. When alarms are detected, the icon changes to the next one.



RPC Server

DDE Server

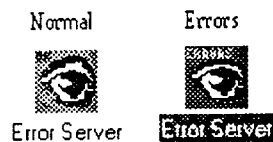
For explanation see the DDE Server program, but replace DDESRV by RPCSRV to establish a DDE link.

If the Simulation Mode is set, the alarms program is stopped and cannot be restarted. In that mode every "RPC" returns a random value between 0 and the Simulation Maximum Value (see RPC Options menu).

3. The Error Server

The Error Server is a program that receives error messages from all the running applications using the RegisterError function. The program normally runs in background and you see a green icon. There are two errorlevels possible :

- low level : the icon becomes red when a message arrives. It stays red until the user activates the server in order to read the message.
- high level : the errorserver is automatically activated and shows the message during 6 seconds. Afterwards it shrinks back to a green icon.



Error Server		
File	Options	Help
DATE-TIME	APPLICATION	ERROR MESSAGE
12-05-13:38	RPC Server	Link not open -> Alarms (BORCH)
12-05-13:38	RPC Server	illegal read/write attempted -> GPS.TARGET (STAG
12-05-13:38	RPC Server	illegal read/write attempted -> GPS.TARGET
12-05-13:38	RPC Server	value out of range -> GPS.TARGET [CCV]
12-05-13:38	RPC Server	value out of range -> GPS.TARGET

To send a message use the RegisterError function.

```
void RegisterError(ApplicationName,ErrorValue,Description,Activate);
char *ApplicationName;
int ErrorValue;
char *Description;
BOOL Activate;
```

The use *on entry* :

ApplicationName - the name of program that sends the message.
 ErrorValue - value corresponding with the errorlist in l:\ecnod\errors.h. If -1, the value to string is suppressed.
 Description - optional description field, put NULL if you don't want to use it.
 The Error Message field in the program is thus the combination of the ErrorValue and Description (ErrorValueString -> Description).
 Activate - TRUE if the Errorserver has to be activated (high level).

There are no return values.

For an example see the RPCSync-example.

4. Access to Excel Databases

In the Isolde Control System, Databases are handled as tables using Microsoft Excel. All Databases are stored in a subdirectory of the H: drive mapped to the isolde/usr:database disk. When a database must be accessible from a program it is saved in *Comma Separated Value* (CSV) format in a filename with the .CSV extension.

For Example, the spreadsheet:

Name	Type	Unit	FecName	EqpModule
GPS.HTROUG	60 KV Rough	Volts	FECSIM	POWS
GPS.HTFINE	60 KV Fine	Volts	FECSIM	POWS
GPS.HTMEAS	60 KV Measure	Volts	FECSIM	POWS
GPS.TARGET	Target	Amps	FECSIM	POWS
GPS.LINE	Line	Amps	FECSIM	POWS
GPS.FILMNT	Filament	Amps	FECSIM	POWS
GPS.ANODE1	Anode 1	Amps	FECSIM	POWS

would be stored in the file example.csv as:

```
Name,Type,Unit,FecName,EqpModule
GPS.HTROUG,60 KV Rough,Volts,FECSIM,POWS
GPS.HTFINE,60 KV Fine,Volts,FECSIM,POWS
GPS.HTMEAS,60 KV Measure,Volts,FECSIM,POWS
GPS.TARGET,Target,Amps,FECSIM,POWS
GPS.LINE,Line,Amps,FECSIM,POWS
GPS.FILMNT,Filament,Amps,FECSIM,POWS
GPS.ANODE1,Anode 1,Amps,FECSIM,POWS
```

A library of C routines to access easily this kind of databases is available.

To use the library, you should:

1) include the L:\database\database.h file

```
#include L:\database\database.h
```

2) add the L:\database\database.lib library in your program list (using Quick C) or in your makefile (using MS C)

Then you can use the functions:

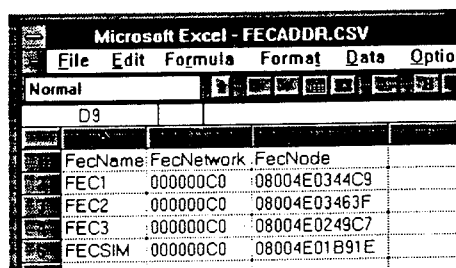
```
int GetDataBaseSize(char *filename);
int ReadDataBase(DataBaseElemsStruct *elems);
int SearchDataBase(DataBaseElemsStruct *elems,char *searchstr,char *colname);
int SearchDataBaseFromFile(char *filename,char *searchstr,char *colname,
                           char *datacol,int datacolien,char *data,int datalen);
int GetDataBaseProperties(char *properties,int size,char *file);
```

to read a .CSV database or to search for a particular value in a table. The structure DataBaseElemsStruct is defined in L:\database\database.h as:

```
typedef struct
{
    char ColName[20];
    long Type;
    void far *Address;
} DataBaseElemsStruct;
```

Return errors, defined in L:\fecnod\errors.h can be not_implemented or no_such_file and are inverted in sign.

Here follows an example to read the H:\EQP\FECADDR.CSV database, from a C program:



FecName	FecNetwork	FecNode
FEC1	000000C0	08004E0344C9
FEC2	000000C0	08004E03463F
FEC3	000000C0	08004E0249C7
FECSIM	000000C0	08004E01B91E

```
#include L:\database\database.h

typedef struct
{
    char Name[RPC_FECNAMELEN+1];
    BYTE Node[6];
    BYTE Network[4];
} RPCFecStruct;

RPCFecStruct far RPCFec[RPC_NUM_FEC];
int RPCNumRecsRead=0;

DataBaseElemsStruct RPCFecElems[]={
    "H:\\EQP\\FECADDR.CSV", sizeof(RPCFecStruct), NULL,
    "FECNAME", STR, RPCFec[0].Name,
    "FECNODE", HEX, RPCFec[0].Node,
    "FECNETWORK", HEX, RPCFec[0].Network,
    ""};

main()
{
    int index;

    RPCNumFecsRead=ReadDataBase(RPCFecElems);

    if (RPCNumFecsRead<0)
        return DATABASE_NOT_LOADED;

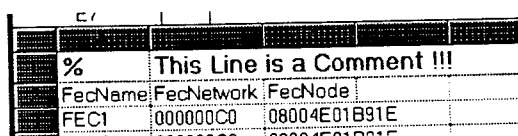
    index=SearchDataBase(RPCFecElems, "FECSIM", "FECNAME");

    printf("FEC named FECSIM is number %d in the FECADDR.CSV database\n", index);

    return 0;
}
```

Supported formats are HEX (hexadecimal), STR (string), INT (integer), FLT (double).

If you want to add comments in your spreadsheet, you must put a % character in column A, as shown in the next figure:



FecName	FecNetwork	FecNode
%	This Line is a Comment !!!	
FEC1	000000C0	08004E01B91E
FEC2	000000C0	08004E0344C9

It is *very important* to use the GetDataBaseSize() function to allocate the memory *before* calling the ReadDataBase() function.

Functions for reading the CSV databases

To read a database you have to define the filename and the columns you want to read.

Each column is an element of the following structure :

```
typedef struct
{
    char ColName[40];
    long Type;
    void far *Address;
}DataBaseElemsStruct;
```

a/ ColName : string that defines the column name in the first active line from the database.

b/ Type : variable type (STR,INT,FLT)

c/ Address : base far address of the array where you want to store the data.

The first element of this structure is an exception and defines the following :

a/ ColName : full path and filename of the database.

b/ Type : size of an element of the memory data structure.

c/ Address : always NULL

Include file "L:\DATABASE\DATABASE.H" into your source and link with the L:\DATABASE\DATABASE.LIB library.

Available Functions :

```
ReadDataBase(DataBaseElemsStruct *DataElems)
```

This reads the database and fills up the memory locations.

The function returns the number of lines read, or -1 if there is an error.

```
SearchDataBase(DataBaseElemsStruct *DataElems, char
*SearchString, char *ColName)
```

After you read the data, it's possible to search in the list by using this function.

Parameters :

SearchString : the string you want to look for.

ColName : the column name in which the function has to search.

The function returns the array index of the string found. It returns -1 if not found.

```
GetDataBaseProperties(char *PropertyList, int Size,
char *FileName)
```

This call provides you a list of all properties of a given equipment module.

Parameters :

PropertyList : two dimensional array of characters (number of properties x stringlength)

Size : Length of one element (stringlength of PropertyList)
 FileName : Full path and name of the equipment module file.

Return value : number of properties read.

Example :

Memory structure :

```
typedef struct
{
    char FecName[20];
    char Name[30];
    char EqpModule[7];
    int EqpNumber;
} DataStruct;

DataStruct far Data[500];

DataBaseElemsStruct DataElems[] = {
    "H:\\SQP\\EQPNAMES.CSV", sizeof(DataStruct), NULL,
    "FECNAME", STR, Data[0].FecName,
    "NAME", STR, Data[0].Name,
    "EQPMODULE", STR, Data[0].EqpModule,
    "EQPNUMBER", INT, &Data[0].EqpNumber,
    ""
};
```

5. Writing Application Programs

There are three supported environments where application programs can be developed. These are *Windows Nodal*, *Microsoft Windows* and *Microsoft Excel*.

Every developer can choose the developing environment according to the performance requirements of his application. Simple applications should be developed using either *Windows Nodal* or *Microsoft Excel*. These two environments provide all the basic tools to access the equipments and to have an user friendly interaction with the user with graphics displays, control panels, buttons. *Microsoft Excel* provides the possibility to build the user interaction using all the *Microsoft Windows* gadgets such as pull down menus, dialog boxes, push buttons, charting and drawing. To use this advanced features of *Excel*, a good knowledge of the *Excel Macro Language* is necessary.

Advanced high performing applications, that require a custom design of the user interface can be developed using the *Microsoft Windows Software Development Kit*. A good knowledge of the C programming Language is necessary.

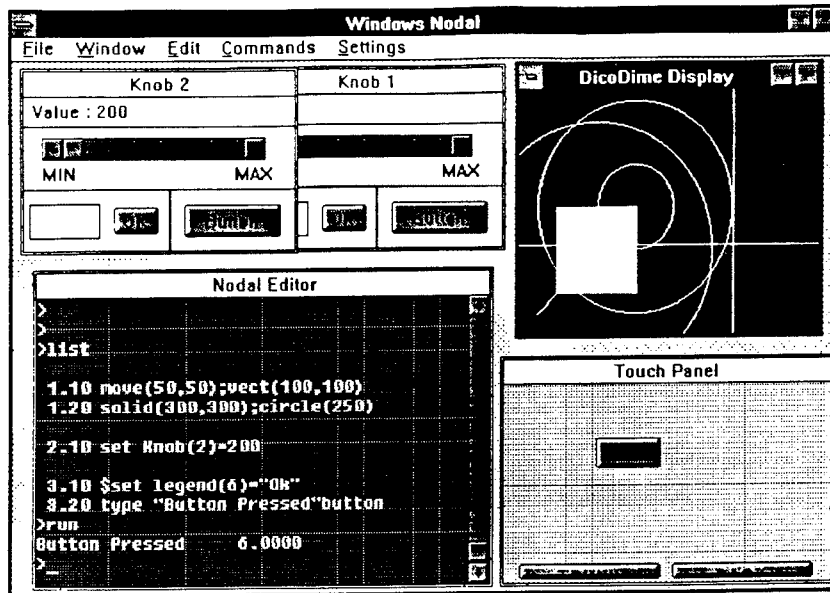
Using Windows Nodal

The *Windows Nodal* Application is a *Nodal* Interpreter that emulates the old *Isolde* consoles.



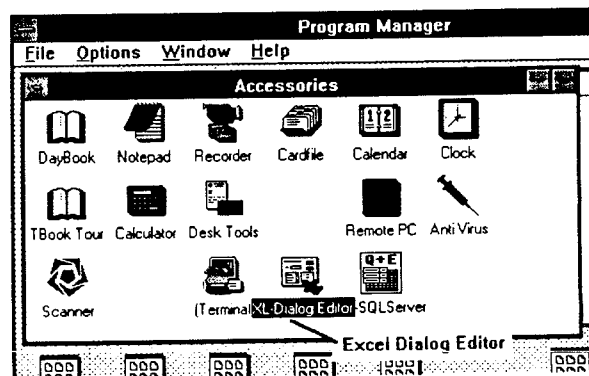
WinNodal

When started, it provides a alphanumeric terminal window, a graphic display window, two knobs and a touch panel window. The manual of the old console to access these devices is still valid. Some new extensions are available from the *nodal* to access the equipments directly from their names instead than from the {FecName, EqModName, EqModNumber} set. This is the *RPC* function. Knobs can be started from the *KnobSrv* with the *KNBCTL* function. Lot of these extensions are not documented and can be found by debugging running *nodal* applications or examples.

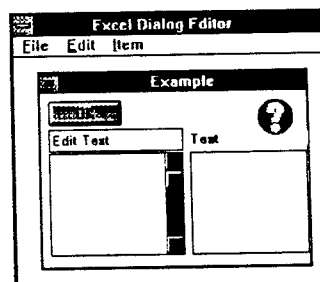


Using Microsoft Excel

All the Excel facilities can be used. This include the Spreadsheet Formulae and the Macros. All features of Excel are documented in the *Excel Reference Book*. To write advanced Macros that uses pulldown menus and dialog boxes, the Excel Dialog Editor should be used. It is available in the *Accessories* program group.

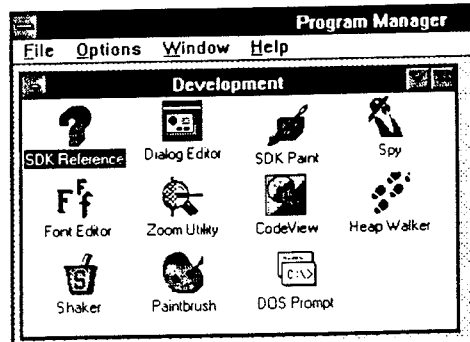


The Excel Dialog Editor allows you to easily create your dialog boxes.

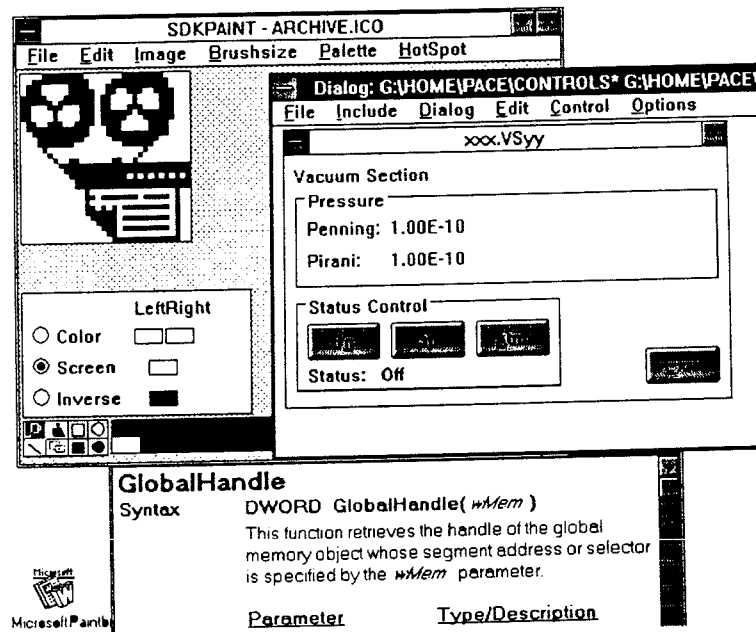


Using the Windows Software Development Kit

To write a Windows application, the Microsoft C compiler and the Windows Software Development Kit are needed. The manuals of both products are necessary. All necessary tools are available in the *Development* program group.



The development all the development products are documented in their manuals and behind the on-line icon named *SDK Reference*.



6. Equipment Module Definition

Equipment Modules (EM) are C Subroutines callable from any PC on the CERN Ethernet in one of the three methods described before (Nodal, C or Windows DDE - Excel). For security reasons, PCs accessing the hardware in Write Mode, must be declared in the console table and be attached to the Isolde Server.

The Equipment Module subroutine is used to access the hardware, and is implemented in the PC that is physically connected with the equipment.

The equipment module file

The equipment module implementation should be done in a file with the same name of the equipment module. For example, the POWP equipment module should be implemented in a file named POWP.C. A template of such a file is l:\eqpmod\dummy.c that defines a dummy equipment module. The Equipment Module file must contain only one subroutine. The name of the subroutine must be equal to the name of the equipment module preceded by a D_

The file must include the l:\fecnod\nodal.h type definition file:

```
#include "l:\fecnod\nodal.h"
```

and the syntax of the equipment module is the following:

```
NodalError D_eqname(real far *value, int flag, int
eqno, char *property)
```

eqname must be replaced by the equipment name.

value is the long address of the double precision floating point value that is received or returned according to *flag*

flag is the read/write flag. Its sign defines if it is a read (positive sign) or a write (negative sign) equipment access. Its absolute value defines the numbers of equipment to read or write and it is generally set to one.

eqno is the equipment number. Remember that an equipment module is handling several homogeneous equipments.

property is the property, dependent on the Equipment module.

The return value of *D_eqname* should be zero when successful or a *NodalError* defined in *nodal.h*

Valid Errors for an Equipment Module are:

```
hardware_error
illegal_equipment_number
illegal_property
illegal_read_write
not_implemented
out_of_range
zero_divide
WA_full
```

The following example shows the implementation of a DEMO equipment module that supports the properties ADDR and CHNN. The file containing this source should be called DEMO.C

```
#include "L:\fecmod\nodal.h"

#define WRITE -1
#define READ 1

typedef struct
{
    real ADDR;
    real CHNN;
} DATA_TABLE;

/* Note: read carefully the chapter on Equipment module initialization to see how the structure Data_Table is initialized
: */

static DATA_TABLE Data_Table[MAX_DEMO]; /* Note: MAX_DEMO is defined in g:fecdef.inc */

NodalError D_Demo(real far *value, int flag, int eqno, char *property)
{
    if (eqno < 1 || eqno > MAX_DEMO)
        return illegal_equipment_number;

    if (!strcmp("ADDR",property))
    {
        if (rwflag==WRITE)
            Data_Table[eqno].ADDR = *value;
        else if (rwflag==READ)
            *value = Data_Table[eqno].ADDR;
        else return illegal_read_write;
    }
    else if (!strcmp("CHNN",property))
    {
        if (rwflag==WRITE)
            Data_Table[eqno].CHNN = *value;
        else if (rwflag==READ)
            *value = Data_Table[eqno].CHNN;
        else return illegal_read_write;
    }
    else
        return illegal_property;

    return 0;
}
```

Note that this equipment module will run on several FECs and the maximum number of equipment (FEC_DEMO) is a FEC-dependent information that has to be included from the g:fecdef.inc file in the g: home directory that should contain, for example:

```
#define FECNAME "FECDEMO"
#define EMFUNS /* to have available the EMINIT function */
#define MAX_DEMO 100
```

Once an equipment module has been developed and is bug-free, it should be moved in the L:\EQPMOD directory, where all the reference Equipment modules are kept, one file per equipment module.

Choosing the Equipment Module Properties

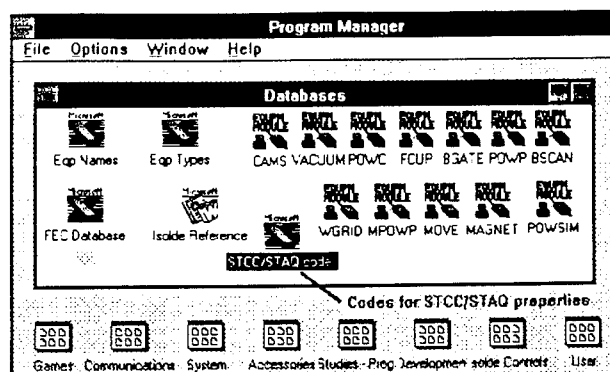
The properties are string *keywords* passed to the equipment module to select a particular function.

All the properties used to write into the equipment module's data table some constants such as the channel number on which every equipment number is connected can be chosen freely. Examples are CHNW or CHNR for the Write or Read channel.

Digital Status Control: STCC and STAQ

If the Equipment Module controls equipments that have a *Status* then the properties to control the status *must* be STCC (control value, read/write property) and STAQ (acquisition, read only property). The meaning of the *value variable passed to the equipment module subroutine is then *coded*

according to the database *STCC/STAQ Codes* available from the *Database* program group.



This database defines all the basic status that are common to all Equipment Modules. For Example the database could contain:

Bit Number	Weight	Description	Comment	STCC	STAQ
0	0	invalid		x	x
1	1	Off		x	x
2	2	On		x	x
3	4	Standby		x	x
4	8	Interlock		x	x
5	16	Local Mode		x	x
6	32	Fault		x	x
7	64	Reset		x	x
8	128	Half	Collimators only	x	x
9	256	Vacuum Status	Vacuum Sections	x	x
10	512	Out		x	x
11	1024	In		x	x

for example, the line

```
SET EQMOD (5, "STCC") =2
```

sets the equipment ON, while the acquisition

```
TYPE EQMOD (5, "STAQ")
10.000
```

tells that the equipment is ON and has an INTERLOCK (8+2 = 10).

Analog Status Control: CCV and AQN

If the Equipment Module controls one analog value, this should be accessible through the CCV (control, read/write property) and AQN (acquisition, read only property) properties.

If the equipment module controls more than one analog values, the *most important* one should be accessible through CCV and AQN while the others with other properties, *starting* with the three CCV and AQN characters, for

examples CCVx and AQNx, where x is a digit (1,2,). A letter can also be used as x but its use is *not* recommended.

ADCs and DACs Scaling Factors

Two additional properties for every analog value (CCVx or AQNx) should be foreseen to set the *Scaling Factor* and the *Scaling Constant*

This properties should be named

SCLW for the DAC Scaling Factor, attached to the CCV property
 SCCW for the DAC Scaling Constant, attached to the CCV property
 SCLR for the ADC Scaling Factor, attached to the AQN property
 SCCR for the ADC Scaling Constant, attached to the AQN property

if CCVx and AQNx properties are used, then SCLWx, SCCWx, SCLRx and SCCRx properties *must* also be defined.

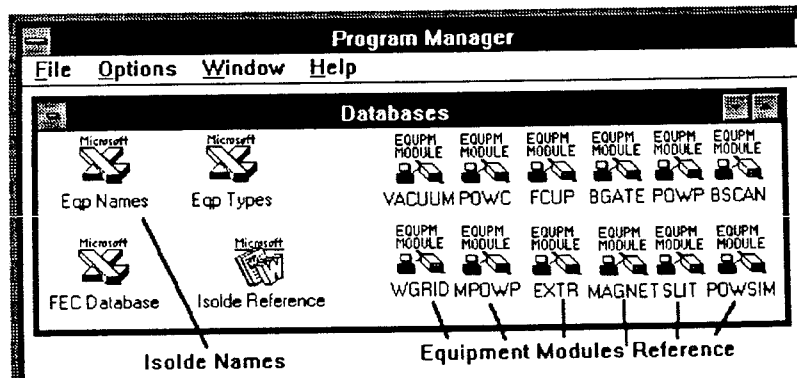
The formula to transform the Value in volts to the ADC or DAC bits is:

$$\text{Value}_{\text{Bits}} = \text{Value}_{\text{PhysicalUnit}} * \text{SCLx} + \text{SCCx}$$

For example a 12 bits DAC has a value in bit ranging from 0 to 4095. If this should be mapped to a current from 0 to 1000 ampère, we would have SCLW=4.095 and SCCW=0. If this should be mapped to a voltage from -100 to 100 volt, we would have SCLW=20.48 and SCCW=2048.

Properties Documentation

All the Equipment Module Properties must be documented by the responsible of the equipment module in its corresponding database file in the *database* program group.



Testing and debugging the Equipment module

To test and debug your equipment module you should:

- 1) put the equipment module file in a dedicated subdirectory in your g:\home directory. For example, if your login name is XYZ, and your equipment module

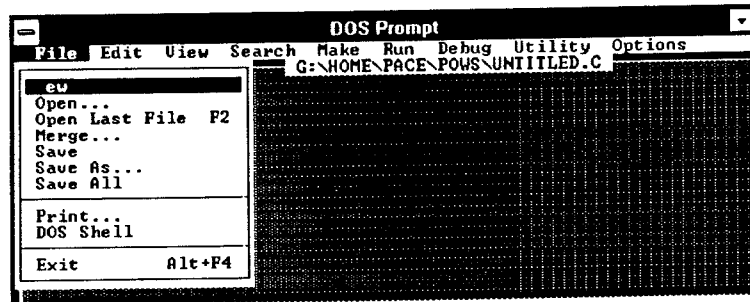
file is called `demo.c`, you should put the `demo.c` file in the `g:\home\xyz\demo` directory.

2) start Quick C with the `QC` command.

3) create, in your `g:\home\yourname\youreqpmod` directory the file `emdecls.inc` by selecting the "New" option in the "File" menu of Quick C. This file should contain the equipment module function prototype in one line, and nothing else. For example in the DEMO equipment module, the prototype of the `D_demo` function is:

```
extern NodalError D_demo(real far *value,int flag,int eqno, char *prop);
```

This file is included during the compilation of the `L:\fecnod\nodfuns.c` file.

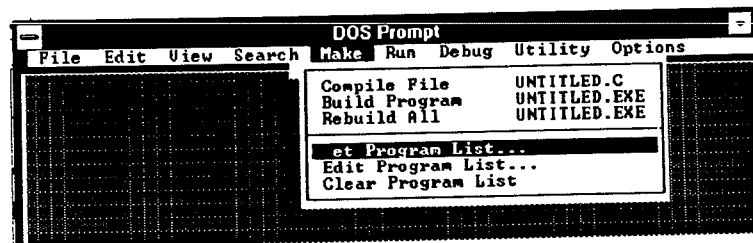


4) create, in your `g:\home\yourname\youreqpmod` directory the file `emspecs.inc` by selecting the "New" option in the "File" menu of Quick C. This file should contain one line, and nothing else. The example for the DEMO equipment module should clarify what you should put in this file.

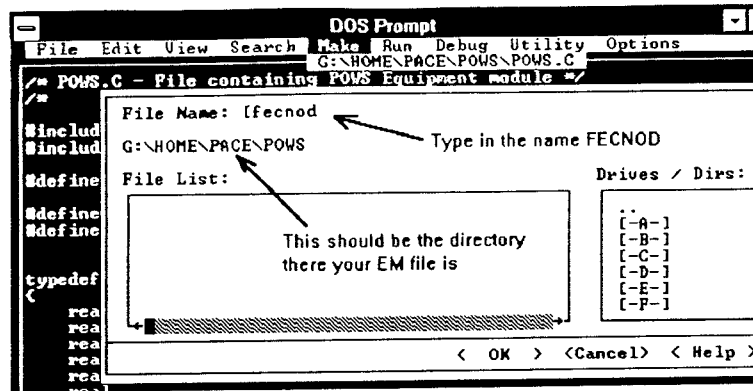
```
data_module,{'D','E','M','O',0,0}, {0,0,0,0}, D_demo,
```

This file is included during the compilation of the `L:\fecnod\nodfuns.c` file.

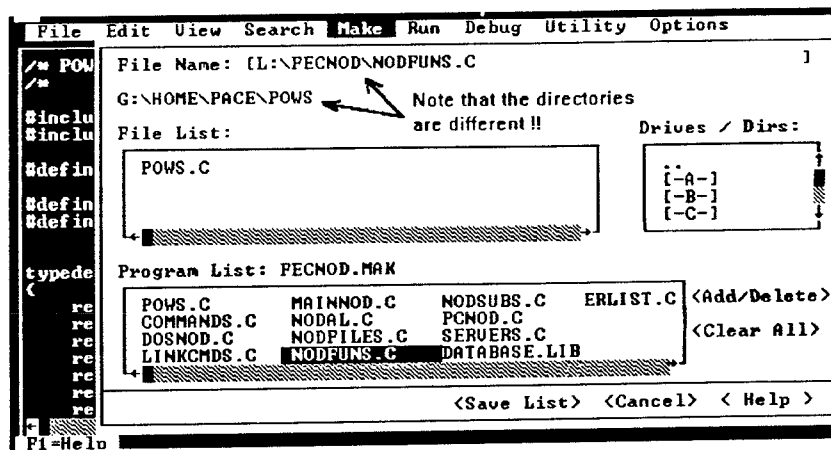
5) create in your `g:\home\yourname\youreqpmod` directory the `fecnod.mak` file by selecting the "Set Program List" option in the "Make" menu of Quick C



You can now create the `fecnod.mak` file:



Then you should set the program list and fills all the files you will need:



The files you need are:

Your Equipment Module File (demo.c or pows.c in the figure).

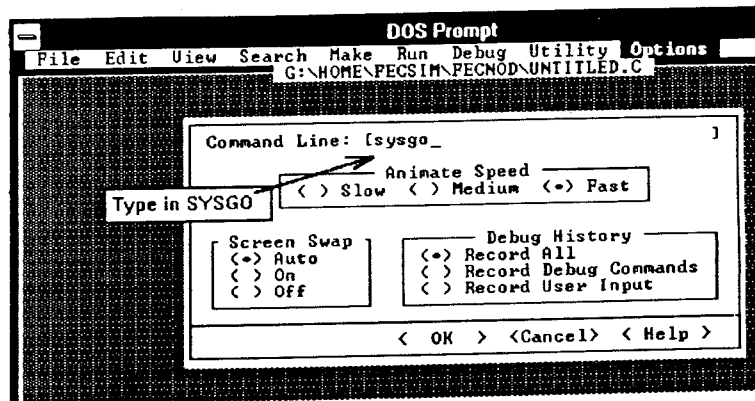
L:\fecnod\commands.c
 L:\fecnod\dosnod.c
 L:\fecnod\erlist.c
 L:\fecnod\linkcmds.c
 L:\fecnod\mainnod.c
 L:\fecnod\nodal.c
 L:\fecnod\nodfiles.c
 L:\fecnod\nodfuns.c
 L:\fecnod\nodsubs.c
 L:\fecnod\pcnod.c
 L:\fecnod\servers.c
 L:\database\database.lib

Now you can click on "Save List" and then select "Rebuild All" in the "Make" menu followed by a "Go" in the "Run" menu.

Equipment Module Initialization

Every Equipment Module has a Data Table that must be initialized. This is achieved by starting the FECNOD program with the SYSGO argument. SYSGO is the name of a Nodal program that will be executed before the network listener is started.

You can type `fecnod sysgo` directly at the DOS prompt or, within Quick C, select "Run / Debug ..." in the "Options" Menu and type in `sysgo`. See next figure.



The `sysgo.nod` nodal program will then initialize all the data table either by calling the equipment module to be initialized for each element number with the different properties or, as recommended, by calling the `EMINIT` nodal function that will call the equipment module several times in order to fill the whole data table.

For example, if the file `H:\EQP\POWS\POWS.CSV` contains the following (excel) database:

EquNo	Name	FecName	MINV	MAXV	SCLR	SCCR	SCLW	SCCW
1	GPS.QP20	FECSIM	0	1000	0.45	0.44	0.23	-0.32
2	GPS.QS30	FECOTH	-100	100	0.11	0.12	0.23	-0.32
2	GPS.DE40	FECSIM	-200	200	0.27	0.34	0.23	-0.32

the nodal line:

```
EMINIT (POWS, "H:\EQP\POWS\POWS.CSV")
```

in the FEC named `FECSIM` is equivalent to the following list of EM calls:

```
set pows (1, "MINV")=0
set pows (1, "MAXV")=1000
set pows (1, "SCLR")=0.45
set pows (1, "SCCR")=0.44
set pows (1, "SCLW")=0.23
set pows (1, "SCCW")=-0.32
set pows (2, "MINV")=-200
set pows (2, "MAXV")=200
set pows (2, "SCLR")=0.27
set pows (2, "SCCR")=0.34
set pows (2, "SCLW")=0.23
set pows (2, "SCCW")=-0.32
```

The calls using "Name" or "FecName" are not generated because they are *mixed case* field. Properties are identified as being in upper case.

Note that the line concerning `GPS.QS30` is ignored because it refers to the initialization of another FEC, namely `FECOTH`.

Properties that should not be initialized can be empty or contain a "x" character. The Empty cell means that the property *is not supported* by that element and the cell with an "x" means that the property *is supported* but does not require an initialization.

6. The Front End Computer

All Front End Computer (FEC) runs the FECNOD program, a NODAL for DOS program that listen to network requests and executes them. The Equipment Module Subroutines are linked with FECNOD.

When the FEC is switched on, it connects to the Isolde Server with a username equal to its name and starts the FECNOD.EXE it finds in its G:\HOME\fecname\FECNOD directory.

A FECNOD program for a FEC is defined by:

- its name
- the list of the Equipment Modules it is handling

When all necessary Equipment module files are installed in the L:\EQPMOD directory, you must ask the Control System Supervisor to:

- 1) Add the FEC in the FEC table if not done yet (h:\eqp\fecaddr.csv)
- 2) If the FEC is new, create an account on the Isolde Server with the Username equal to the FEC Name.
- 3) Create the EM Table list for that particular FEC and (g:\home\fecname\fecnod\emdecls.inc and g:\home\fecname\fecnod\emspecs.inc) and fill the parameters to dimension the Equipment Module Data Tables (g:\home\fecname\fecnod\fecdef.inc).
- 4) create the makefile to compile and link the FECNOD for that FEC. The make file must be named FEC (no extensions) and must be in the G:\HOME\fecname\FECNOD directory.

Then, when logging into the Isolde Server with the username equal to the Fec Name, the FECNOD for that FEC will be automatically regenerated every time it is not up to date.

The emdecls.inc file

This file is included by NODFUNS.C and contains the function prototypes of all equipment modules needed.

Example of the emdecls.inc file for a FEC with two EM, namely POWS and FCUP:

```
extern NodalError D_pows(real far *value,int flag,int eqno, char *prop);
extern NodalError D_fcup(real far *value,int flag,int eqno, char *prop);
```

The emspecs.inc file

This file is included by NODFUNS.C and contains part of the functions translation table.

The fecdef.inc file

```
#define EMFUNS
#define ALARMS
#define SERVER
#define CLIENT
```

Example of the fecdef.inc file for a FEC called "FECSIM" with two EM, namely POWS and FCUP. The FEC has 20 equipments on POWS and 15 on FCUP

```
#define FECNAME "FECSIM"
#define EMFUNS
#define MAX_POWS 20
#define MAX_FCUP 15
```

This file is passed to the MAKE utility to generate the FECNOD, at FEC startup. The fecnod is, of course, regenerated only when some modifications occurred in the source files.

The FECNOD programs is supposed to be generated with the Microsoft C v6.0 or later and *not* with the Quick C Compiler.

```
cc = cl -c -AMD -G2s -Ocegit -Wl -FPi87
nodpath = 1:/fecnod/
eqppath = 1:/eqpmod/
home = g:

commands.obj: $(nodpath)commands.c $(nodpath)nodal.h fec.
               $(cc) $(nodpath)commands.c

dosnod.obj: $(nodpath)dosnod.c $(nodpath)nodal.h fec.
             $(cc) $(nodpath)dosnod.c

linkcmds.obj: $(nodpath)linkcmds.c $(nodpath)nodal.h fec.
               $(cc) $(nodpath)linkcmds.c

mainnod.obj: $(nodpath)mainnod.c $(nodpath)nodal.h fec.
              $(cc) $(nodpath)mainnod.c

nodal.obj: $(nodpath)nodal.c $(nodpath)nodal.h fec.
            $(cc) $(nodpath)nodal.c
```

```
nodfiles.obj: $(nodpath)nodfiles.c $(nodpath)nodal.h fec.  
$(cc) $(nodpath)nodfiles.c  
nodfuns.obj: $(nodpath)nodfuns.c $(nodpath)nodal.h $(home)emdecis.inc $(home)emspecs.inc $(home)fecdef.inc fec.  
$(cc) $(nodpath)nodfuns.c  
nodsubs.obj: $(nodpath)nodsubs.c $(nodpath)nodal.h fec.  
$(cc) $(nodpath)nodsubs.c  
pcnod.obj: $(nodpath)pcnod.c $(nodpath)nodal.h fec.  
$(cc) $(nodpath)pcnod.c  
servers.obj: $(nodpath)servers.c $(nodpath)nodal.h $(nodpath)novell.h fec.  
$(cc) $(nodpath)servers.c  
fcup.obj: $(eqppath)fcup\fcup.c $(nodpath)nodal.h $(home)fecdef.inc fec.  
$(cc) $(eqppath)fcup\fcup.c  
pows.obj: $(eqppath)pows\pows.c $(nodpath)nodal.h $(home)fecdef.inc fec.  
$(cc) $(eqppath)pows\pows.c  
fecnod.exe: commands.obj dosnod.obj linkcmds.obj mainnod.obj nodal.obj nodfiles.obj nodfuns.obj nodsubs.obj pcnod.obj  
servers.obj pows.obj fec. linklist  
link @linklist
```

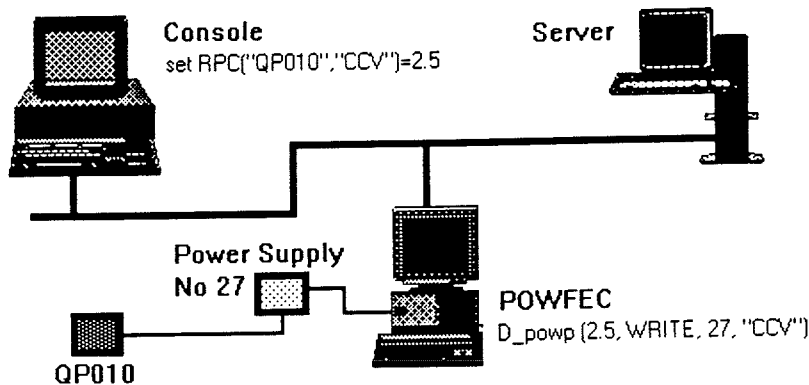
8. Network Call translation

This small chapter explains you how an RPC call is translated into an Equipment Module Call.

In reference to the following picture, let us take an example of a

```
set rpc("QP010","CCV")=2.5
```

command given to the Nodal for Windows interpreter.



The "QP010" Name is decoded into the element database (h:\eqp\eqpnames.csv) and as a results, are found:

The FEC Name (POWFEC) of the FEC that is controlling the QP010 element .
 The Equipment Module Name (POWP) that is controlling QP010 on that FEC.
 The Equipment Number (27) of QP010 in POWP.

The FEC Address is then extracted from the h:\eqp\fecaddr.csv database from its name (POWFEC) and a RPC call is sent to the correct FEC, specifying the Equipment Module to call (POWP) and the Equipment Number (27), the property (CCV), the Read/write Flag (WRITE) and the Value (2.5).

The Network listener of the FECNOD programs, running in POWFEC, receives the messages and executes it by calling the POWP subroutine with all parameters received. All the parameters plus an eventual return value and a completion code are then sent back to the client console when the execution of the POWP subroutine has terminated.

The RPC is a synchronous call because it waits the equipment module to finish execution. Asynchronous call can be obtained with different properties, one to trigger the measurement, one to check if the measurement is finished and a third one to read the measure.

The eqpnames and fecaddr databases are read once, at the beginning of program execution and then kept in RAM (of the client console). The search in these databases to retrieve the FEC name, EM name and equipment number is extremely fast and give to the application program a complete independence

from these latter. The only hard-coded things inside the application programs are logical names and properties of the control system.

9. Isolde Names

An Isolde Name is a character string up to 10 characters that defines *uniquely* an element of the control system that can be accessed.

Element Names and Equipment Modules properties (see later) are hard-coded into applications programs and can therefore *never be modified* once baptized. This is an important point to keep in mind when deciding on new names.

Definition:

The **first three characters** of the name define the zone in which the element is. Valid zones are:

GPS - for the General Purpose Separator
 GLM - GPS Low mass side of beams
 GHM - GPS High mass side of beams
 HRS - for the High Resolution Separator
 MSW - Merging Switch Yard, for vacuum only.

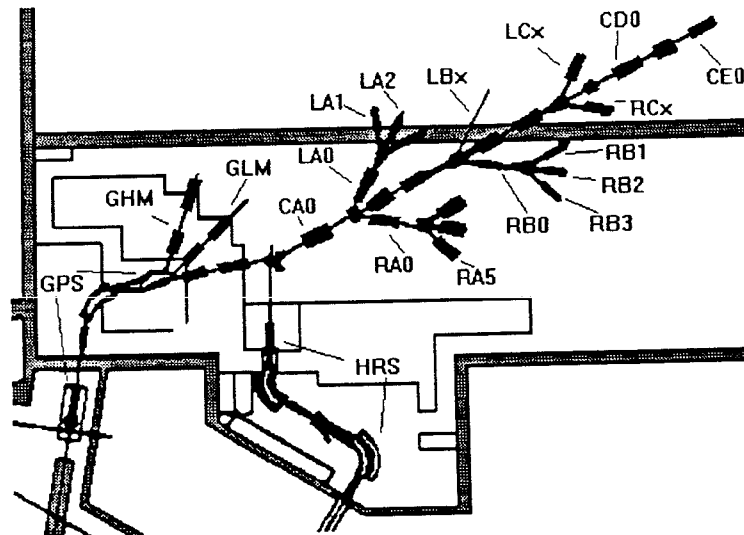
In the Experimental hall, the three character zones are defined in the following way:

The first character is C for the central beam line, otherwise L for the Left side and R for the Right side of the beam line.

The second character can be A, B, C, D or E and identifies the different sections counted from the merging switchyard.

The third character is a digit indicating the different branches of the section.

Examples of sections are shown in the next picture.



The **fourth character** is a period (.).

The **fifth and sixth characters** describes the element types. These are:

QP - Quadrupole
 QS - Quadrupole Steering

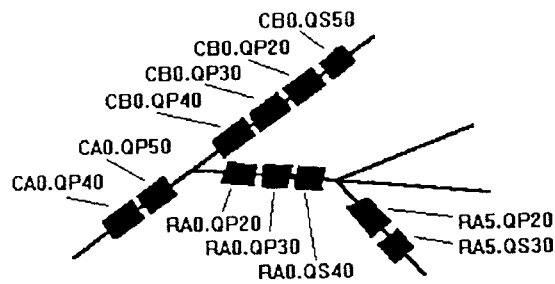
MP - Multipole
 GR - Guard Ring
 XY - X-Y Correction Plate
 BE - Bender
 KI - Kicker
 DE - Deflector Electrical
 DP - Deflector Position
 FC - Faraday Cup
 UC - User Cup (Measurement Point)
 SC - Scanner
 SP - Scanner Position
 WG - Wire Grid
 SL - Slit
 LC - Lens Collimator
 BG - Beam Gate
 VS - Vacuum Section
 VV - Vacuum Valve in the beam line

The last two or three characters are a three digit number. The first digit represents a sub-zone of the main zone. The second digit and the third digit are a sequence number. In the design phase, elements will be named in step of 10 to allow insertion of new elements. At the machine startup, the last digit of all names should always be a zero.

All elements that have well known names that identify them, can have a **unique 6 character** name preceded by the zone name and the period (.).

Example of this last names are: HRS.FILMNT, GPS.HTMEAS.

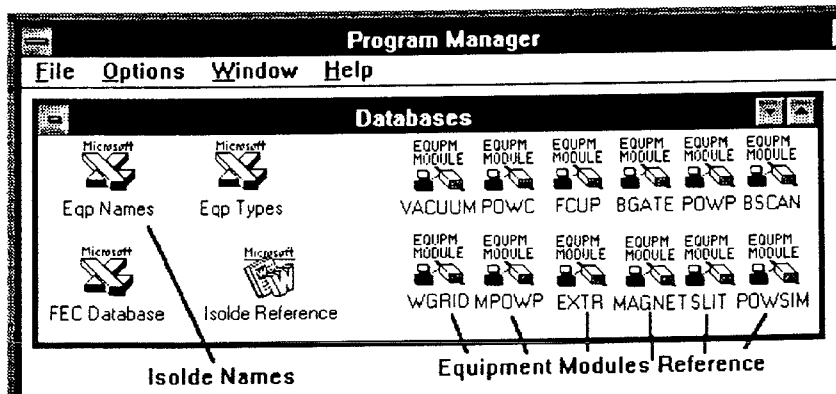
Example of names following the general rule are shown in the next figure.



Annex: Equipment Module List and Element Database

Here you can find the Equipment modules defined up to now, together with all Equipments and their names.

As this document, experience has shown that equipment modules have a certain tendency to become obsolete rapidly. You can at any moment read the latest version by clicking on the icon of your Equipment Module in the *Database* program Group of the Isolde Console on the Isolde Server. From the PS Network you can read this document by issuing a login `isolde/guest` command to the dos prompt.



Databases can be easily printed on the printer of your choice that can be selected with the applications *Network Printers (Winspool)* and *Control Panel*, as shown in the introduction.

Example of an Equipment Module listing

```
/* POWSIM.C - File containing POWSIM Equipment module */
/* - Used to Simulate POW functions */

#include "stdlib.h"
#include "L:/FECNOD/nodal.h"

#define WRITE -1
#define READ 1

typedef struct
{
    real CCV;
    real MINV;
    real MAXV;
    real SCLR;
    real SCCR;
    real SCLW;
    real SCCW;
    real STCC;
    real STAQ;
} DATA_TABLE;

static DATA_TABLE DataTable[MaxPOWSIM]; /* Note: the Initialization Values should disappear !! */

NodalError D_powsim(real far *Value, int RWFlag, int EqNo, char *Property)
{
    int Eq;
    /*
    printf("POWSIM: eqp no = %d, Property = %s, flag = %d", EqNo, Property, RWFlag);
    */
    if (EqNo < 1 || EqNo > MaxPOWSIM) {
        printf("\a\nILLEGAL EQUIPMENT NUMBER\n");
        return illegal_equipment_number;
    }
    Eq = EqNo - 1;

    if (!strcmp(Property, "CCV")) {
        switch(RWFlag) {
            case WRITE:
```

```

        if( *Value < DataTable[Eq].MINV || *Value > DataTable[Eq].MAXV ) {
            printf("\a\nVALUE OUT OF RANGE - %lf\n", *Value);
            return out_of_range;
        }
        DataTable[Eq].CCV = *Value;
        break;
    case READ:
        *Value = DataTable[Eq].CCV;
        break;
    default:
        printf("\a\nILLEGAL READ WRITE ATTEMPTED\n");
        return illegal_read_write;
}

else if (!strcmp(Property,"MINV")) {
    switch(RWFlag) {
        case WRITE:
            DataTable[Eq].MINV = *Value;
            break;
        case READ:
            *Value = DataTable[Eq].MINV;
            break;
        default:
            printf("\a\nILLEGAL READ WRITE ATTEMPTED\n");
            return illegal_read_write;
    }
}

else if (!strcmp(Property,"MAXV")) {
    switch(RWFlag) {
        case WRITE:
            DataTable[Eq].MAXV = *Value;
            break;
        case READ:
            *Value = DataTable[Eq].MAXV;
            break;
        default:
            printf("\a\nILLEGAL READ WRITE ATTEMPTED\n");
            return illegal_read_write;
    }
}

else if (!strcmp(Property,"SCLR")) {
    switch(RWFlag) {
        case WRITE:
            DataTable[Eq].SCLR = *Value;
            break;
        case READ:
            *Value = DataTable[Eq].SCLR;
            break;
        default:
            printf("\a\nILLEGAL READ WRITE ATTEMPTED\n");
            return illegal_read_write;
    }
}

else if (!strcmp(Property,"SCCR")) {
    switch(RWFlag) {
        case WRITE:
            DataTable[Eq].SCCR = *Value;
            break;
        case READ:
            *Value = DataTable[Eq].SCCR;
            break;
        default:
            printf("\a\nILLEGAL READ WRITE ATTEMPTED\n");
            return illegal_read_write;
    }
}

else if (!strcmp(Property,"SCLW")) {
    switch(RWFlag) {
        case WRITE:
            DataTable[Eq].SCLW = *Value;
            break;
        case READ:
            *Value = DataTable[Eq].SCLW;
            break;
        default:
            printf("\a\nILLEGAL READ WRITE ATTEMPTED\n");
            return illegal_read_write;
    }
}

else if (!strcmp(Property,"SCCW")) {
    switch(RWFlag) {
        case WRITE:
            DataTable[Eq].SCCW = *Value;
            break;
        case READ:
            *Value = DataTable[Eq].SCCW;
            break;
        default:
            printf("\a\nILLEGAL READ WRITE ATTEMPTED\n");
            return illegal_read_write;
    }
}

else if (!strcmp(Property,"AQN")) {
    switch(RWFlag) {
        case READ:
            *Value = DataTable[Eq].CCV * (1 + (((real) rand())-32000.)/330000);
            break;
        default:
            printf("\a\nILLEGAL READ WRITE ATTEMPTED\n");
            return illegal_read_write;
    }
}

else if (!strcmp(Property,"STCC")) {
    switch(RWFlag) {
        case WRITE:
            if( *Value < 0 || *Value > 64 ) {
                printf("\a\nVALUE OUT OF RANGE - %lf\n", *Value);
                return out_of_range;
            }

```

```

        }
        DataTable[Eq].STCC = *Value;
        break;
    case READ:
        *Value = DataTable[Eq].STCC;
        break;
    default:
        printf("\a\nILLEGAL READ WRITE ATTEMPTED\n");
        return illegal_read_write;
    }
}
else if (!strcmp(Property, "STAG")) {
    switch(RWFlag) {
        case READ:
            *Value = DataTable[Eq].STCC;
            break;
        default:
            printf("\a\nILLEGAL READ WRITE ATTEMPTED\n");
            return illegal_read_write;
        }
    }
else {
    printf("\a\nILLEGAL PROPERTY\n");
    return illegal_property;
}
}
/*
*/
return 0;
}

```

Appendix F : Colored figures.

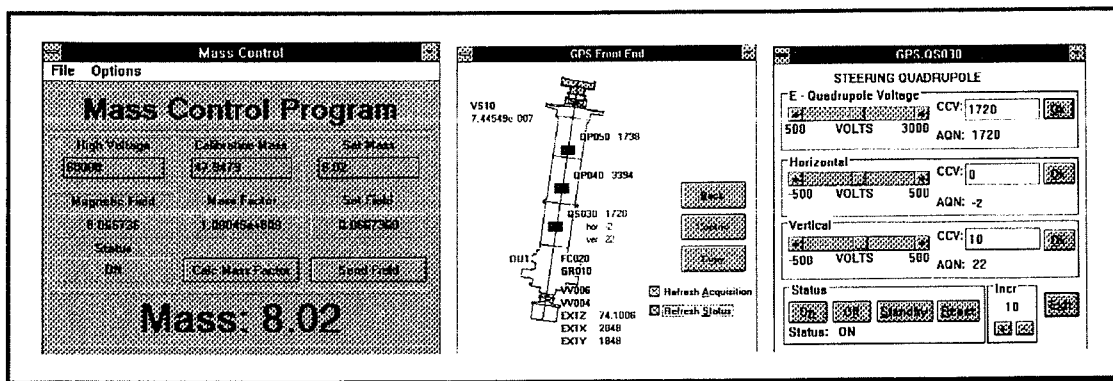


Figure 4.3 Example, from the left to the right, of an application, a synoptic and a knob.

Error Server		
File	Options	Help
DATE-TIME	APPLICATION	ERROR MESSAGE
09-29-11:23	RPC Server	Non existent Element -> Poke GPS.TEST_CCV
09-29-11:23	RPC Server	Non existent Element -> Control Panel GPS.TEST
09-29-11:23	RPC Server	Non existent Element -> Control Panel GPS.TEST
09-29-11:23	RPC Server	Non existent Element -> Control Panel GPS.TEST
09-29-11:23	RPC Server	Non existent Element -> Reading Panel GPS.TEST_CCV 0

Figure 4.4. The Error Server.

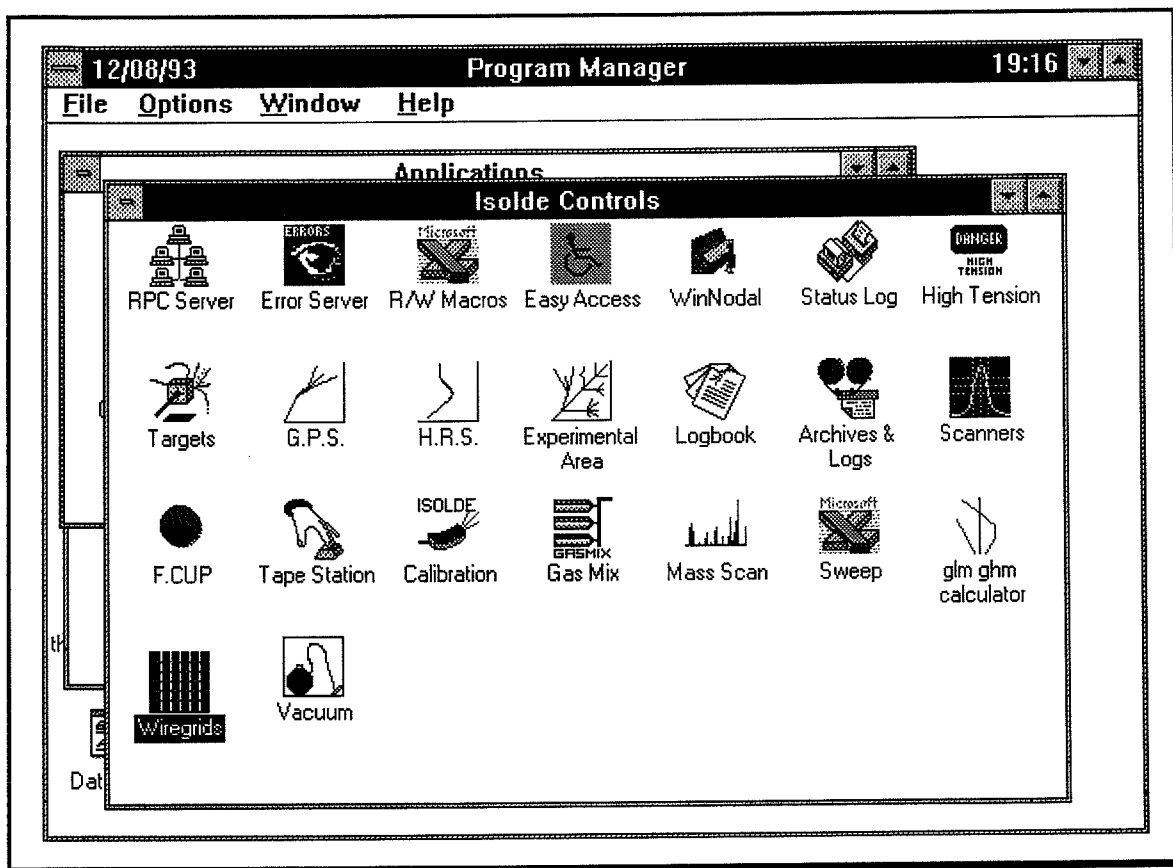


Figure 4.6. The Graphic User Interface.

Microsoft Excel									
File Edit Formula Format Data Options Macro Window Help									
Normal									
A1									
MPOWP.CSV									
	A	B	C	D	E	F	G	H	
1	%	Database for MPOWP Equipment Module							
2	EqpNumb	Name	FecName	MINV	MAXV	MINV2	MAXV2	MINV3	M
3	1	GPS.QS58	POWER1	500	3000	-500	500	-500	
4	2	GLM.QS70	POWER1	500	3000	-500	500	-500	
5	3	GHM.QS71	POWER1	500	3000	-500	500	-500	
MPOWPDOC.XLS									
	A	B	C	D	E	F	G	H	
2	MPOWP Multiple Value Power Supplies access for the								
3									
4		Analog (Standard)							
5			INIT	R/W	ADCCurrent the same on all poles				
6			CCV	R/W	ADC Current in Amps				
7			CCVD	R/W	ADC in Bits				
8			AQN	R	DAC Current in Amps				
9			AQND	R	DAC in Bits				
10			CCV2	R/W	ADC Current in Amps				
Ready									

Figure 4.7. The Excel Database.

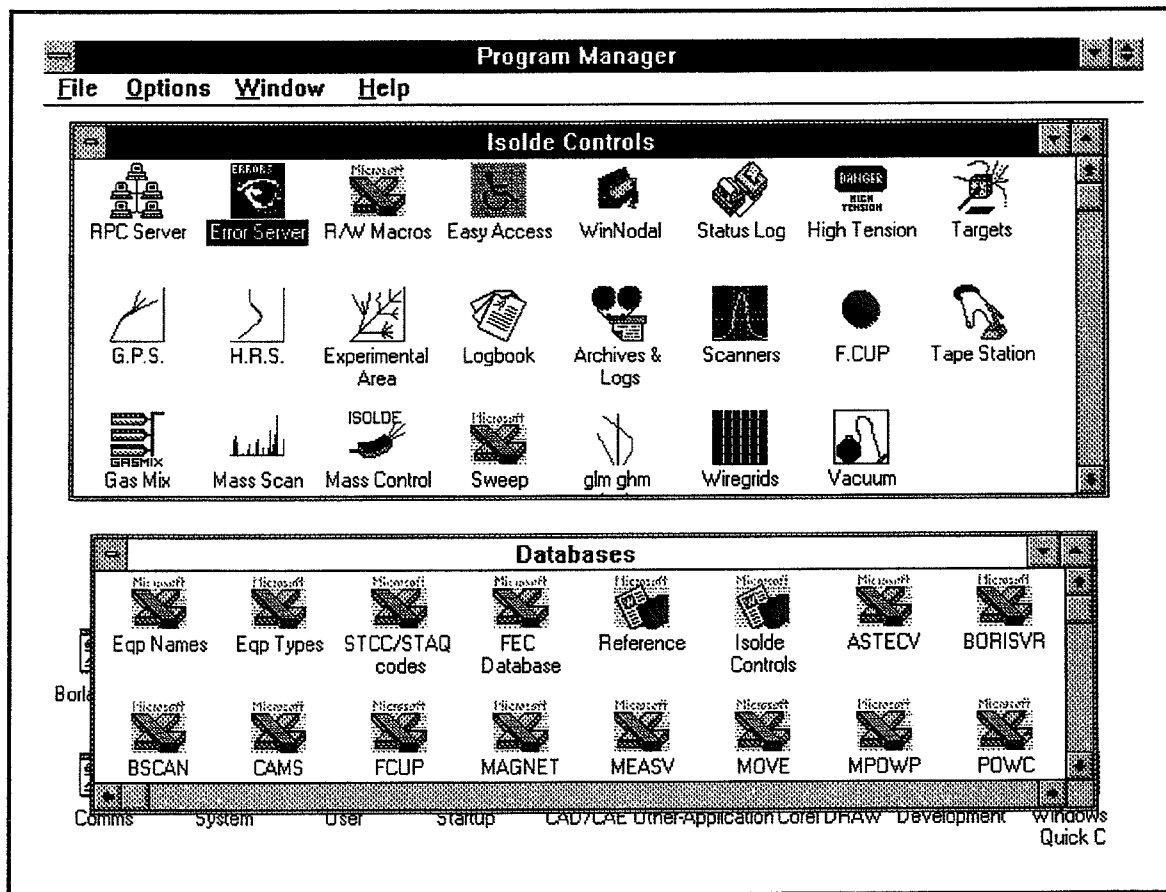


Figure 5.1. The ISOLDE Controls and Databases groups.

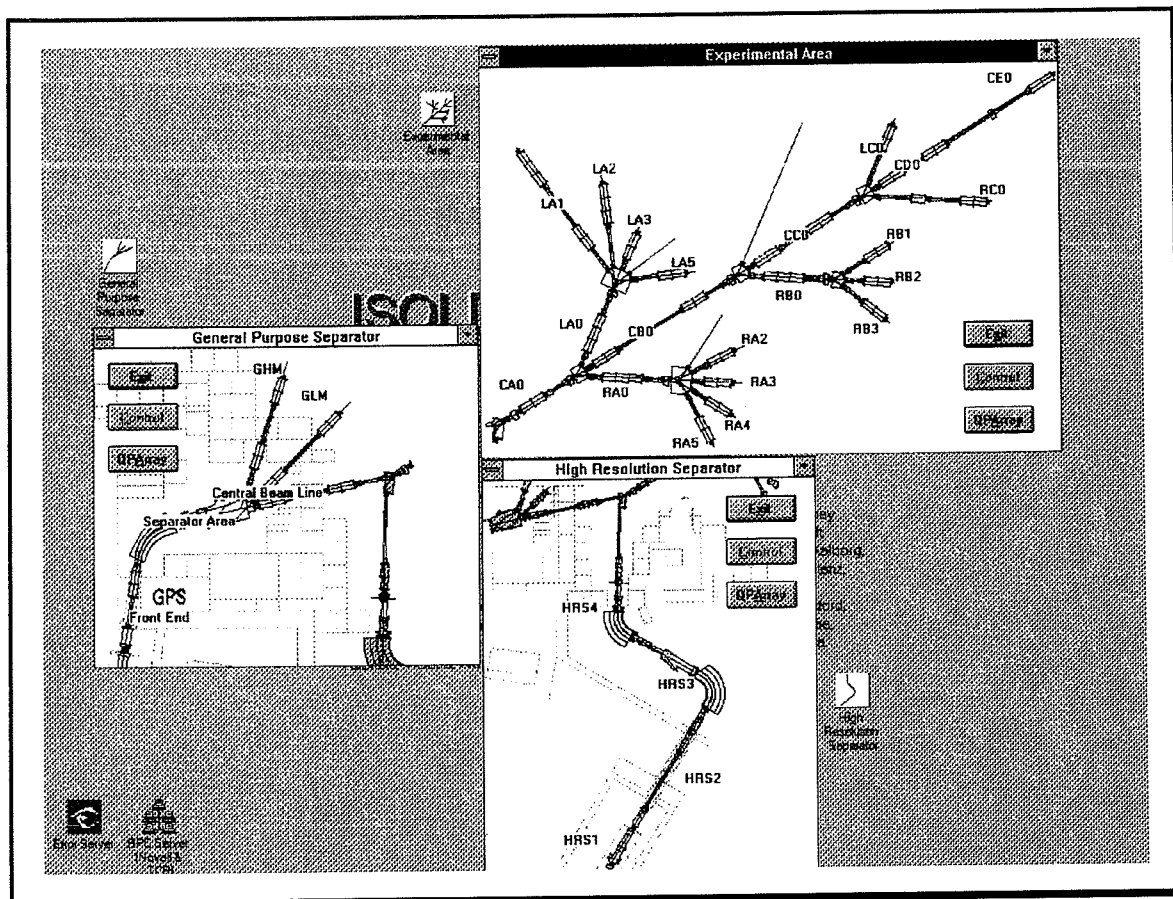


Figure 5.2. Complete view of the beam network and related icons.

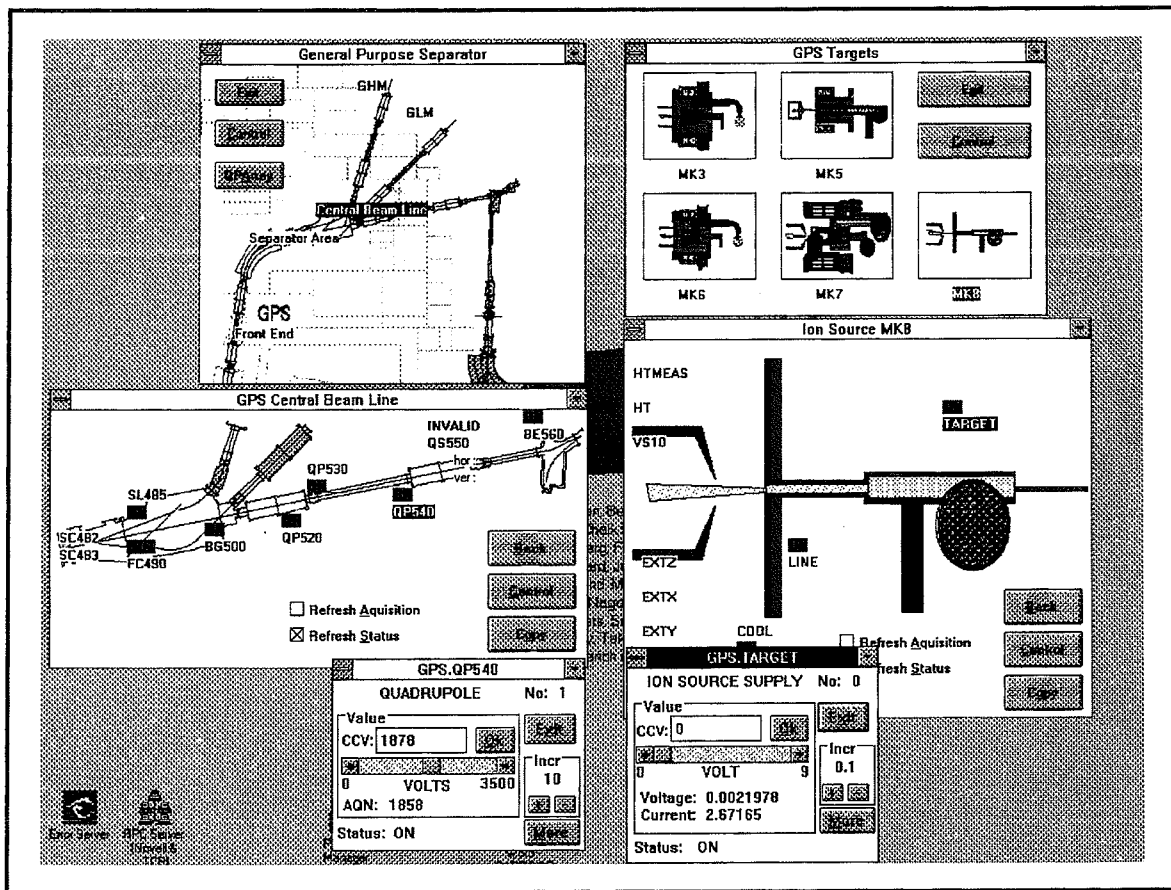


Figure 5.3. On two vertical columns are two applications with the same internal structure.

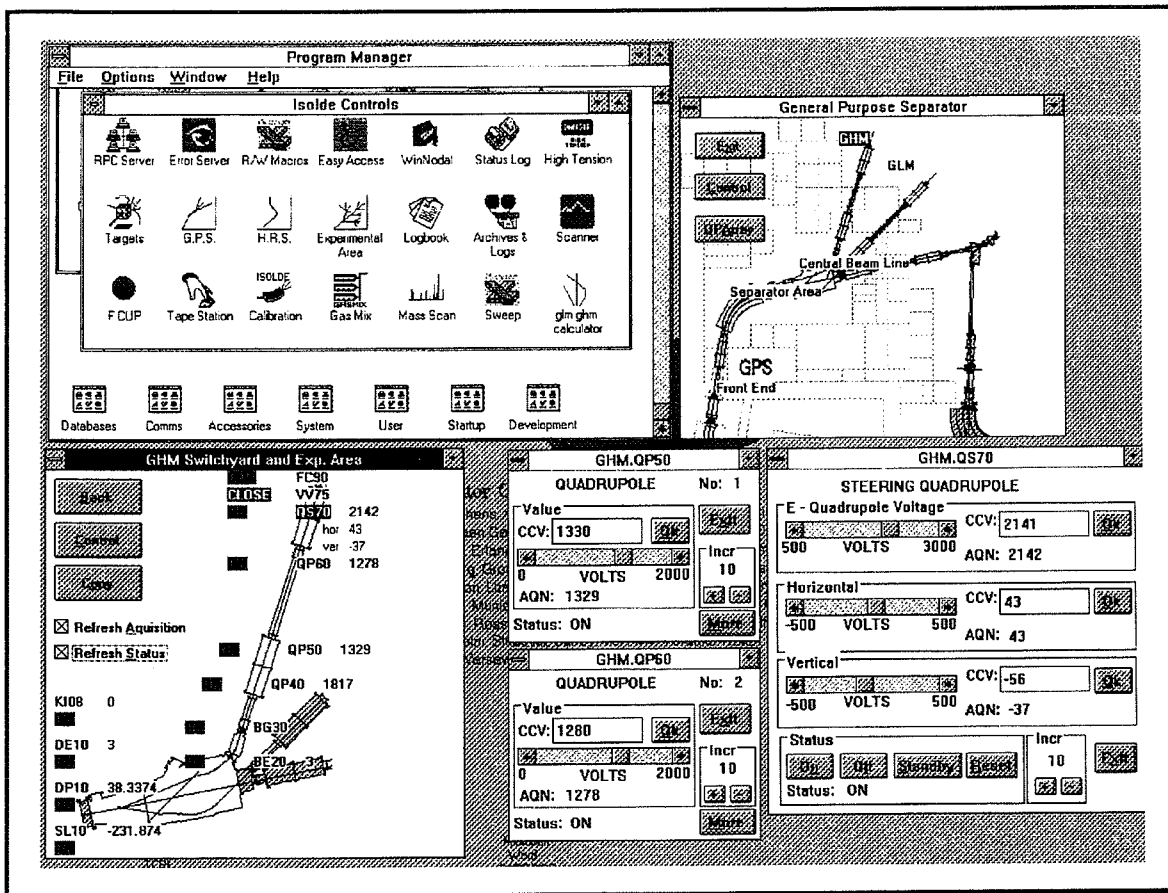


Figure 5.4. Typical screen representation, with the screen divided into four main areas.

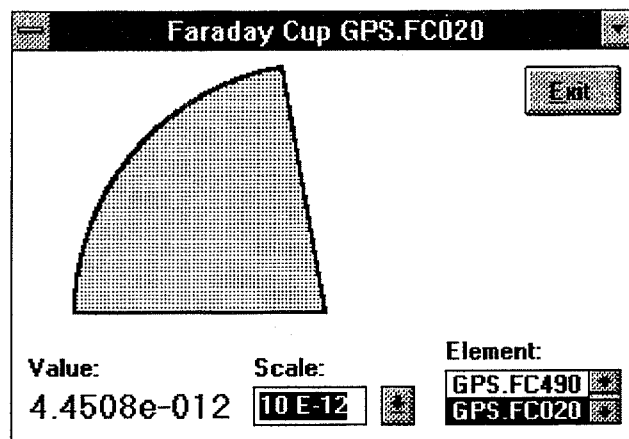


Figure 5.6. The Faraday Cup Application.

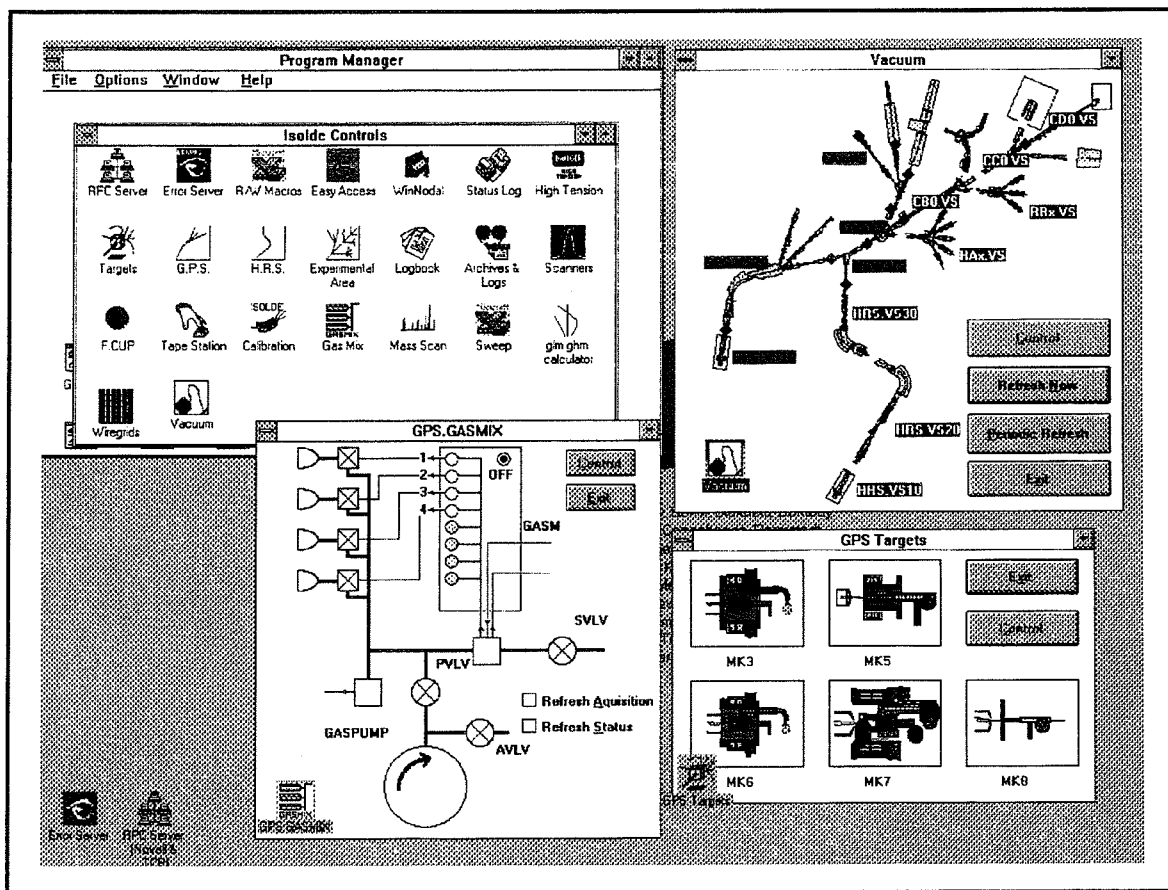


Figure 5.7. Photographically real imagery has been used for synoptics, GPS gasmix and GPS targets.

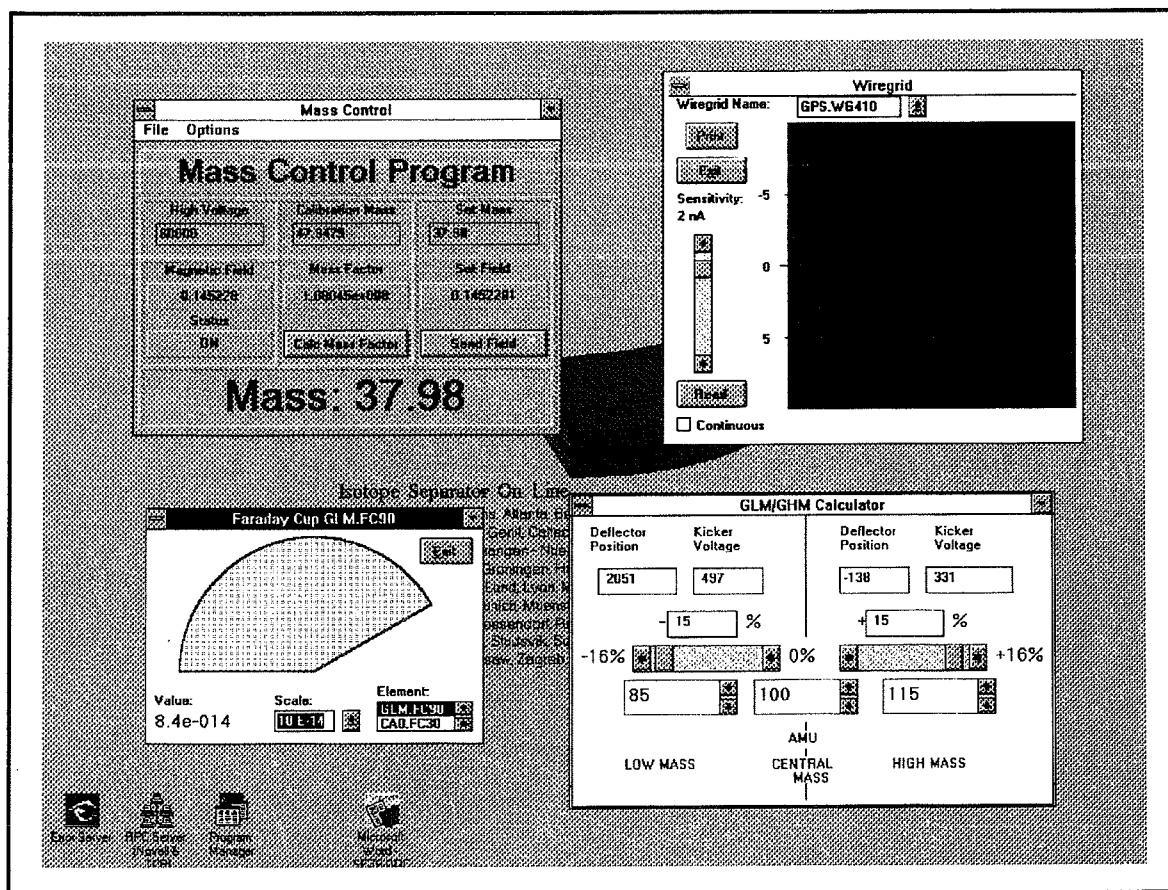


Figure 5.8. Abstract imagery was chosen for applications such as Mass Control, Wiregrid, Faraday Cup and GLM/GHM Calculator.

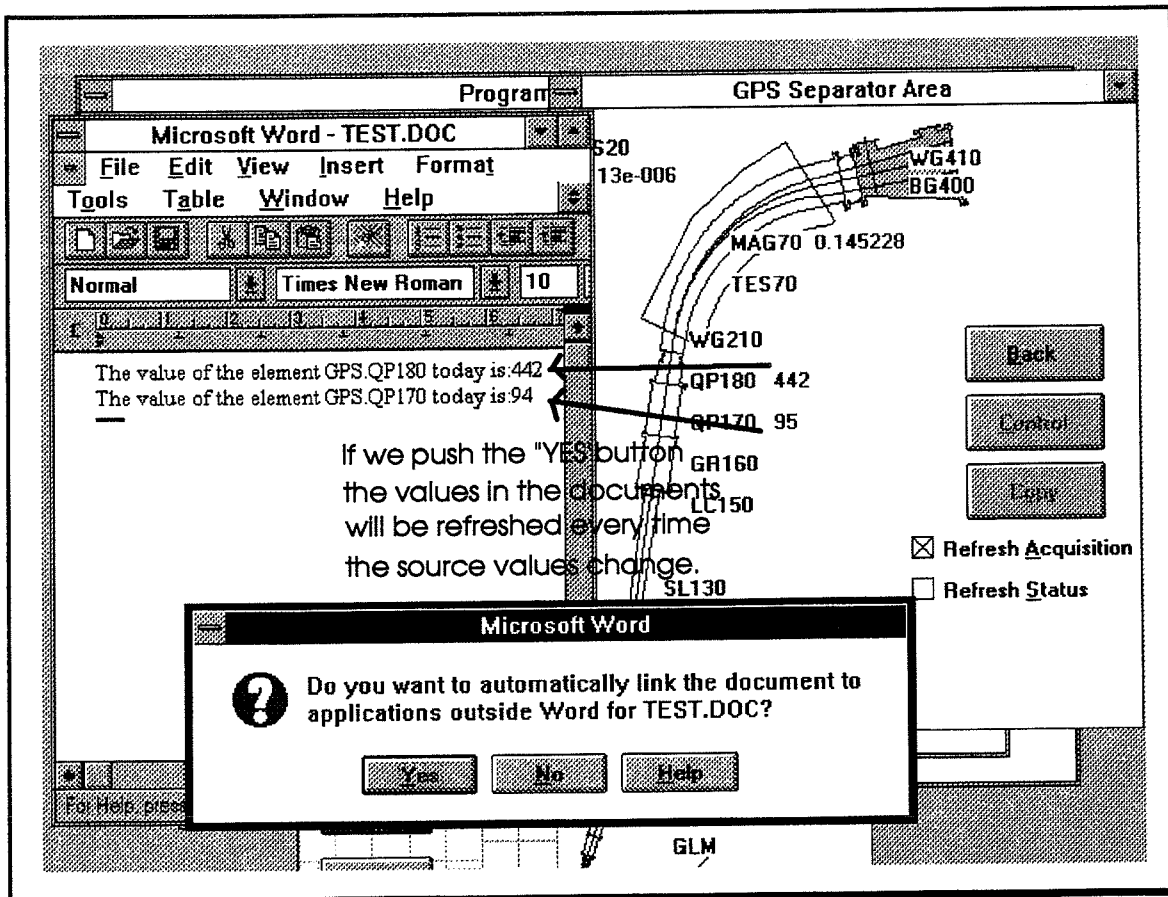


Figure 5.9. When we try to open the Word file TEST.DOC we are asked if we want the linked values to be refreshed if the source changes.

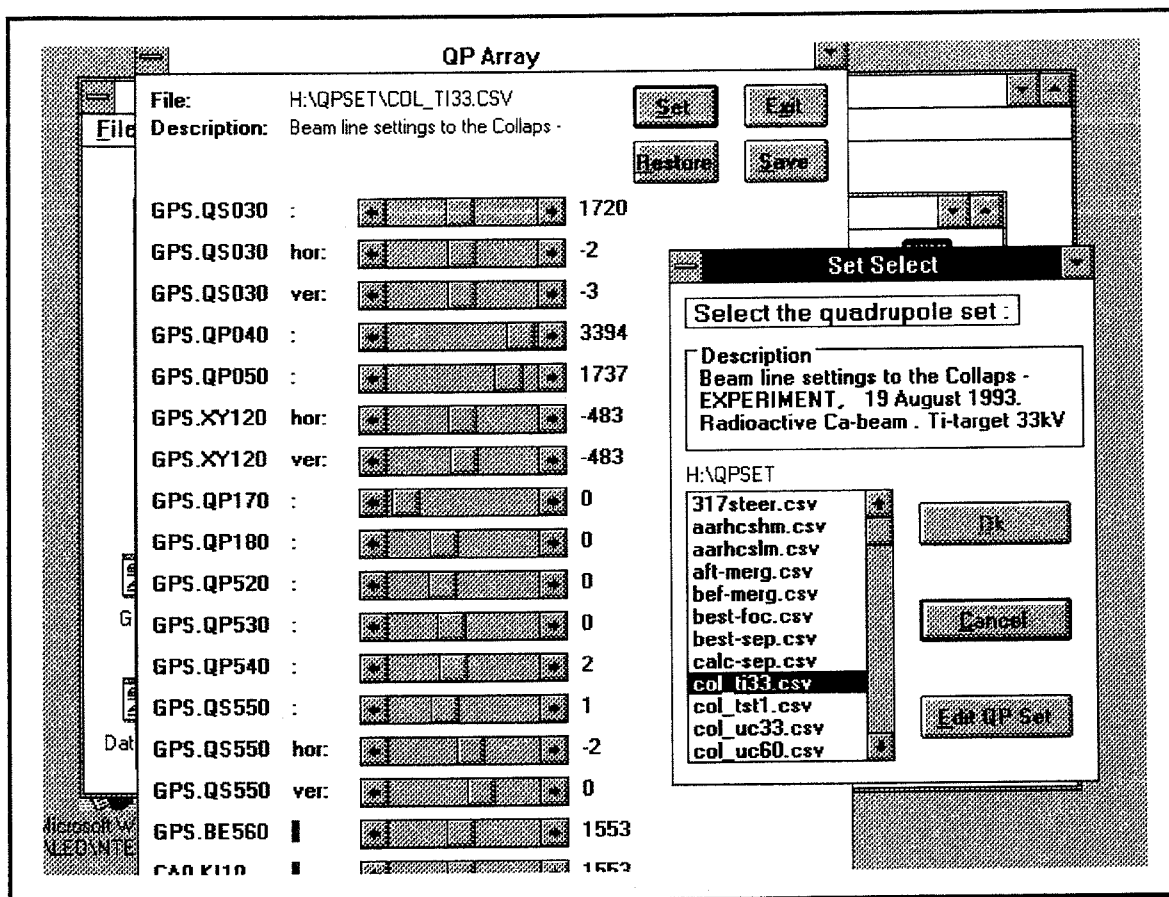


Figure 5.10. The set select window of the QP-Array application permits to choose the set of elements to control. The QP-Array windows permits to modify, store and retrieve configuration values.

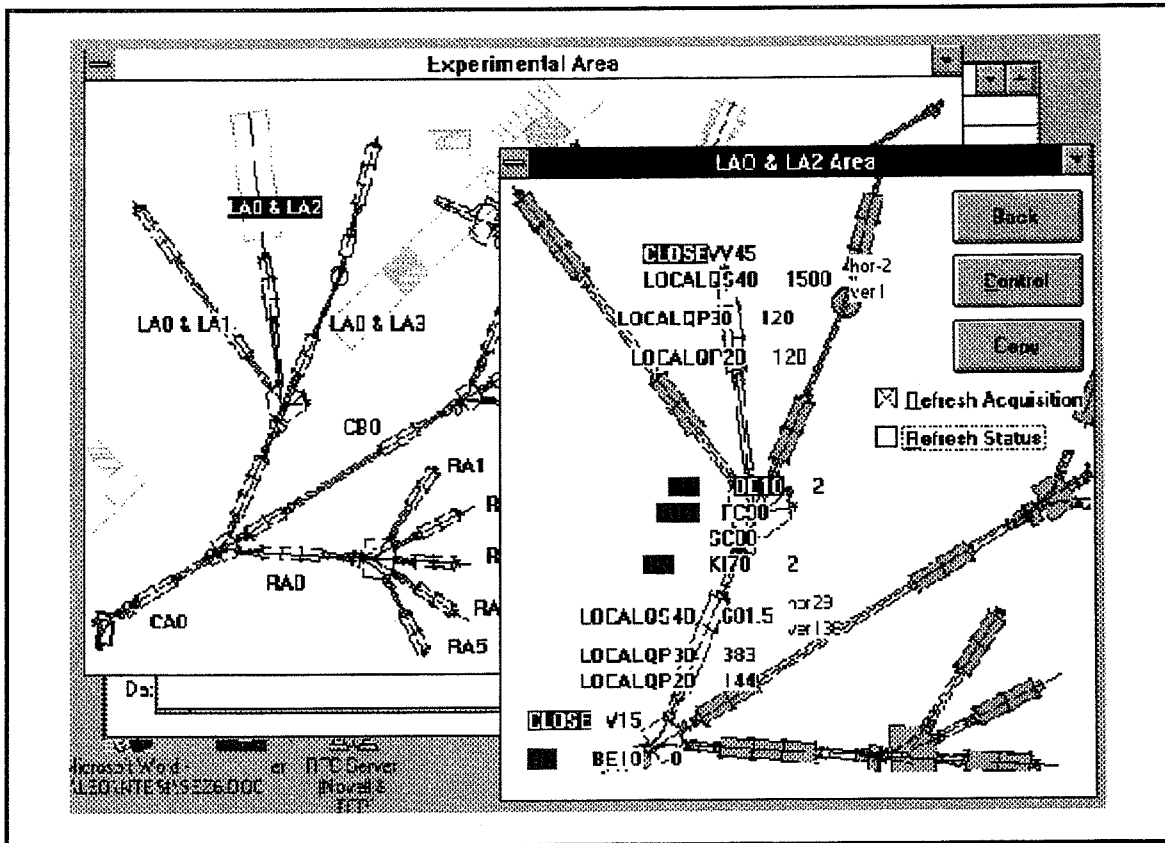


Figure 5.11. Although the information relay on text the color code help to convey information.

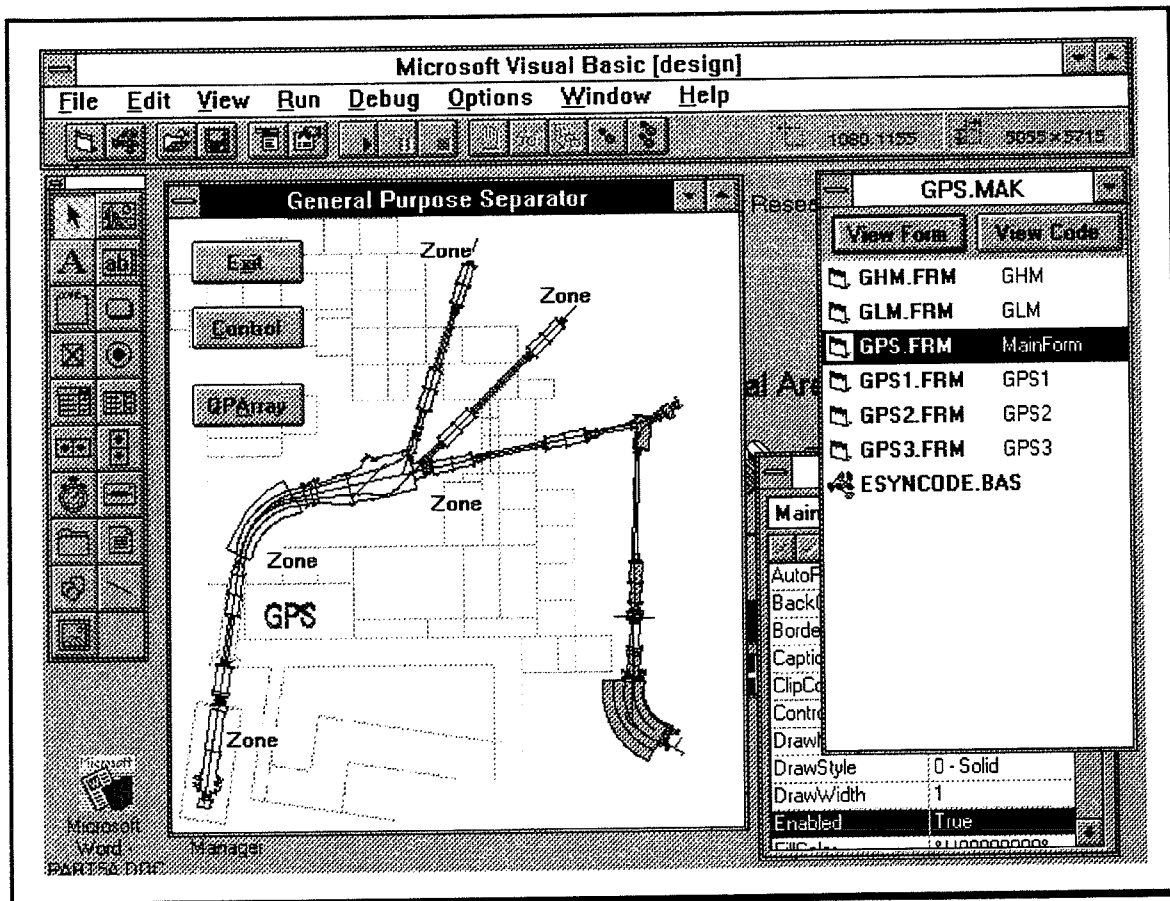


Figure 5.12. The Visual Basic™ development environment.

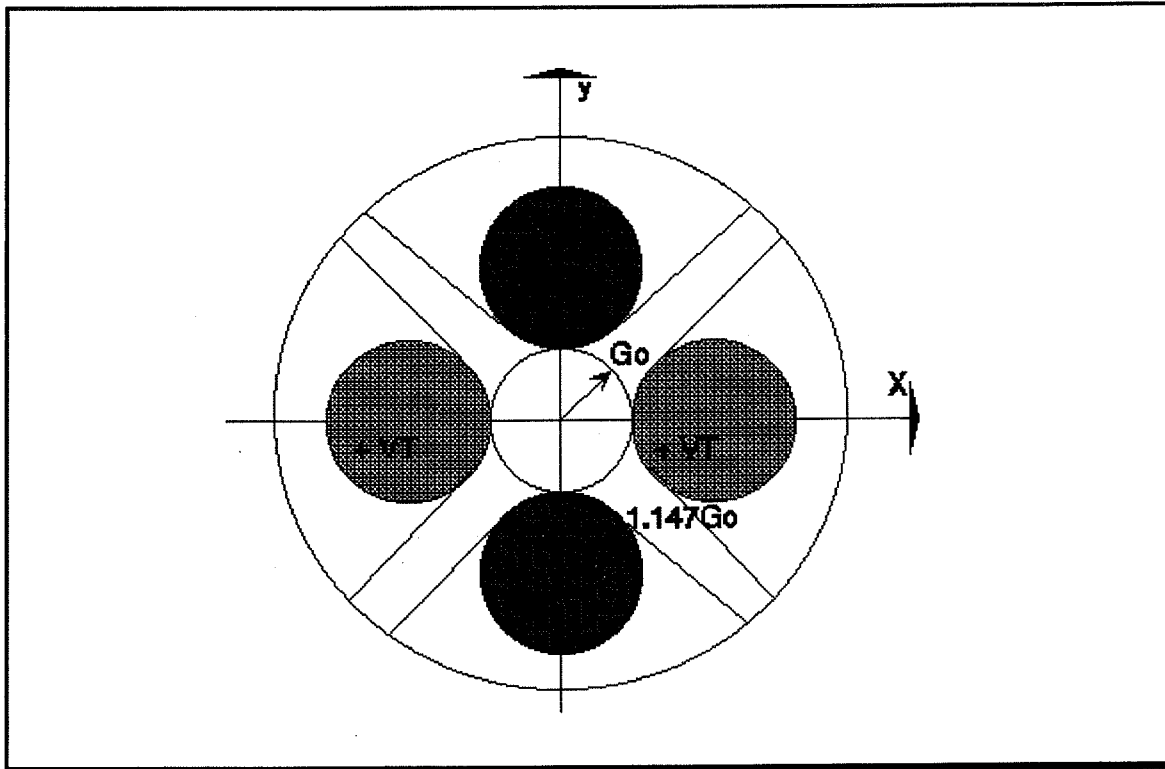


Figure 6.2. Approximation of hyperbolic electrodes with circular ones.

	S	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
	Name	FacName	MINV	MAXV	SCLR	SCOR	SCLW	SCOW	ADRR	ADRW	CHNR	CHNW	CTRW	ADRD	CHND	CCV	AON	STAQ
1	GPS.OP140	POWER1	0	3500	1.17	0	1.17	0:100h	160h		2	2	100h		52	0:x	x	
2	GPS.OP150	POWER1	0	2000	2.0475	0	2.0475	0:100h	160h		3	3	100h		52	0:x	x	
3	GPS.OP170	POWER1	0	2000	2.0475	0	2.0475	0:100h	160h		4	4	1079bh	100h	54	0:x	x	
4	GPS.OP180	POWER1	0	1250	3.2768	0	3.2768	0:100h	160h		5	5	100h		54	0:x	x	
5	GPS.OP520	POWER1	0	3500	1.17	0	1.17	0:100h	160h		6	6	100h		54	0:x	x	
6	GPS.OP530	POWER1	0	3500	1.17	0	1.17	0:100h	160h		7	7	10b9bh	100h	56	0:x	x	

Figure 6.5. Database of some of the QPs.

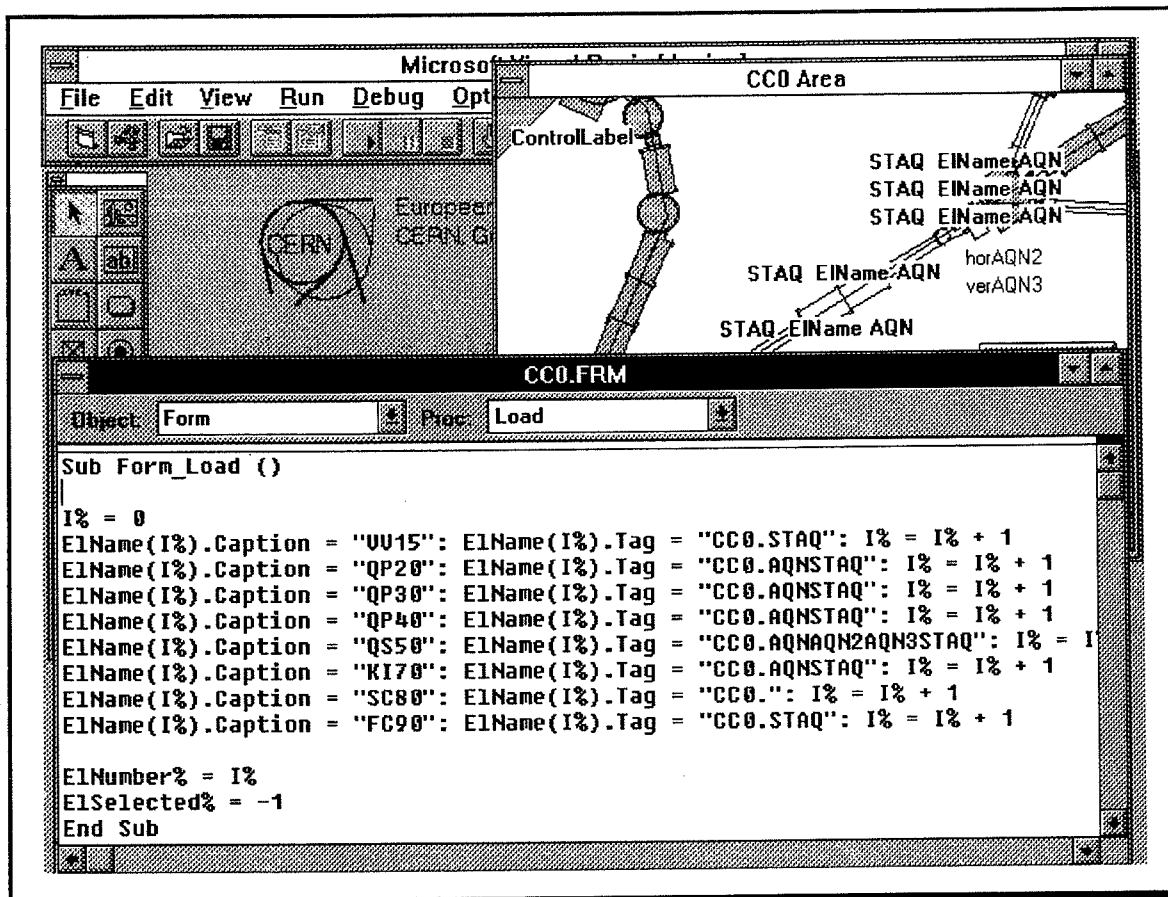


FIGURE 7.2 The load procedure of a Visual Basic™ application.

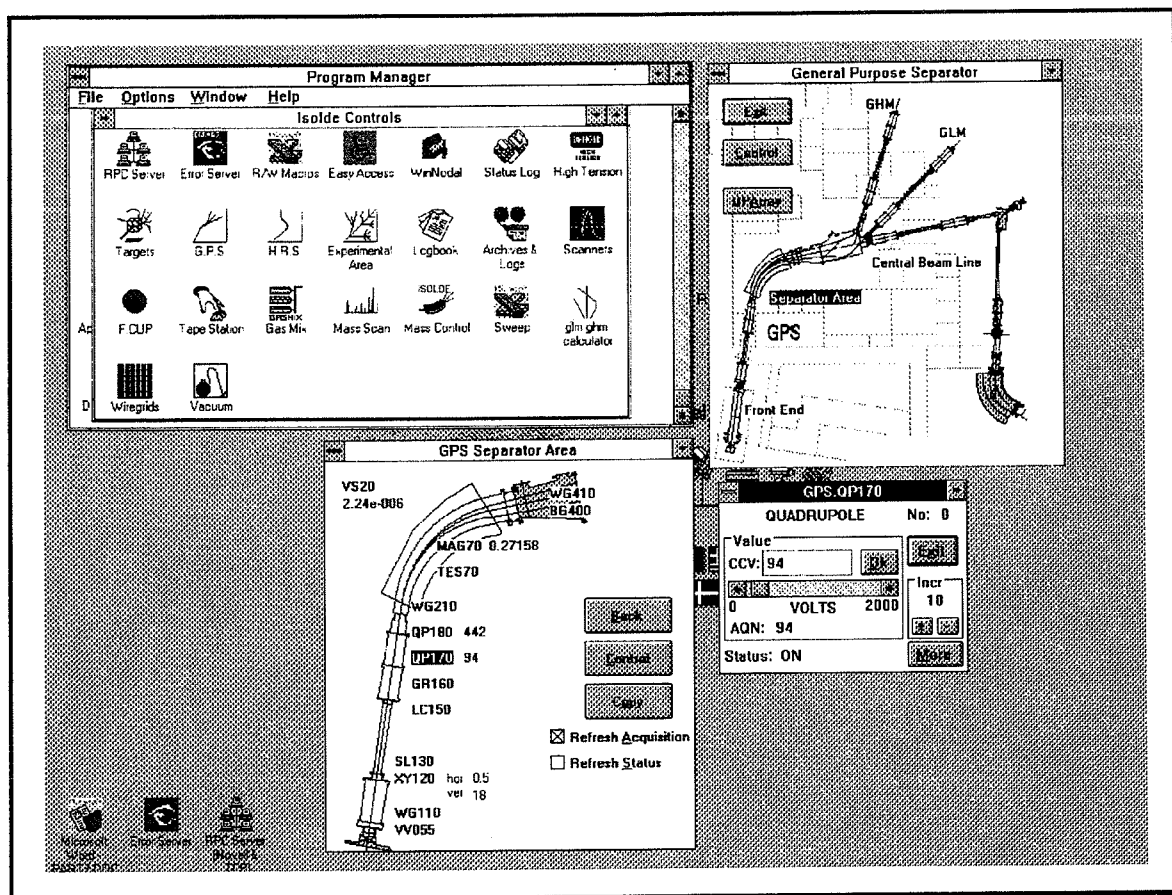


Figure 7.3. The GPS application used to control the GPS.QP170 QP.

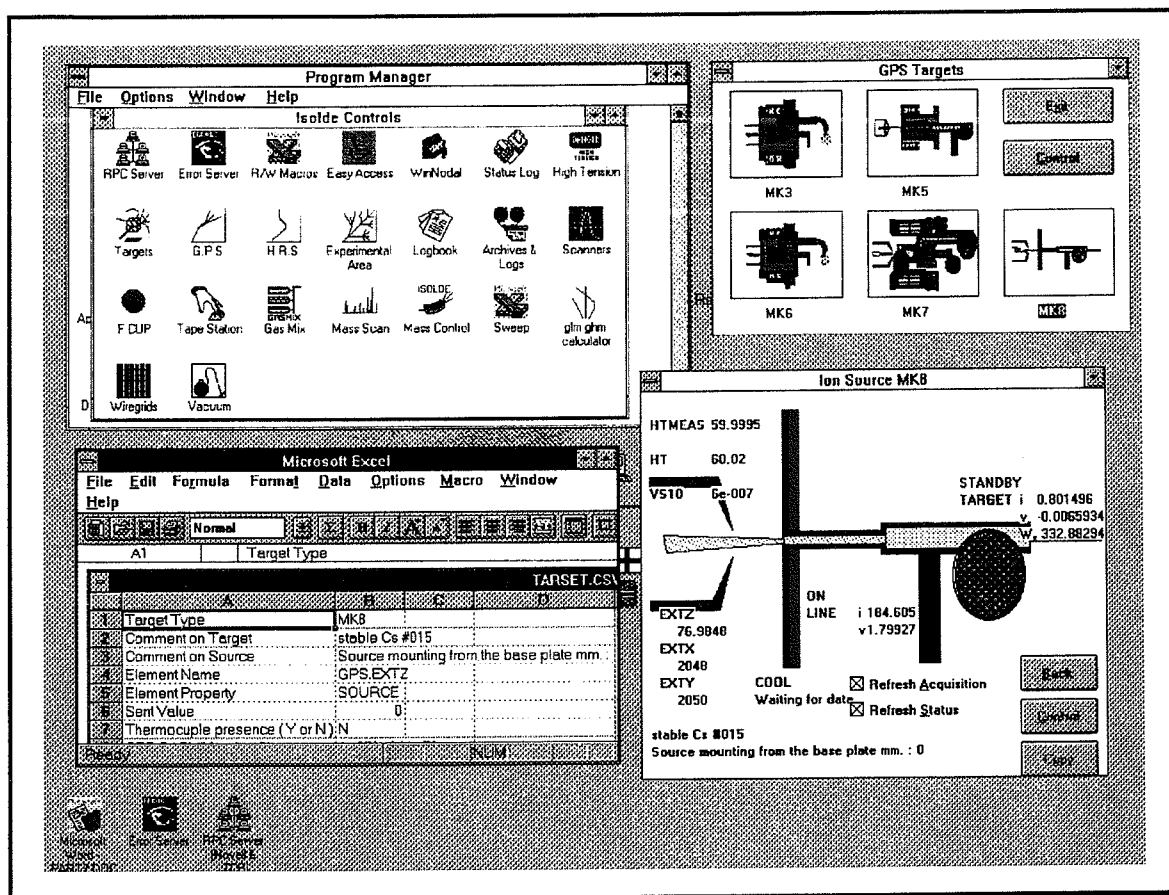


Figure 7.4. The Targets application and the database file TARSET.CSV.

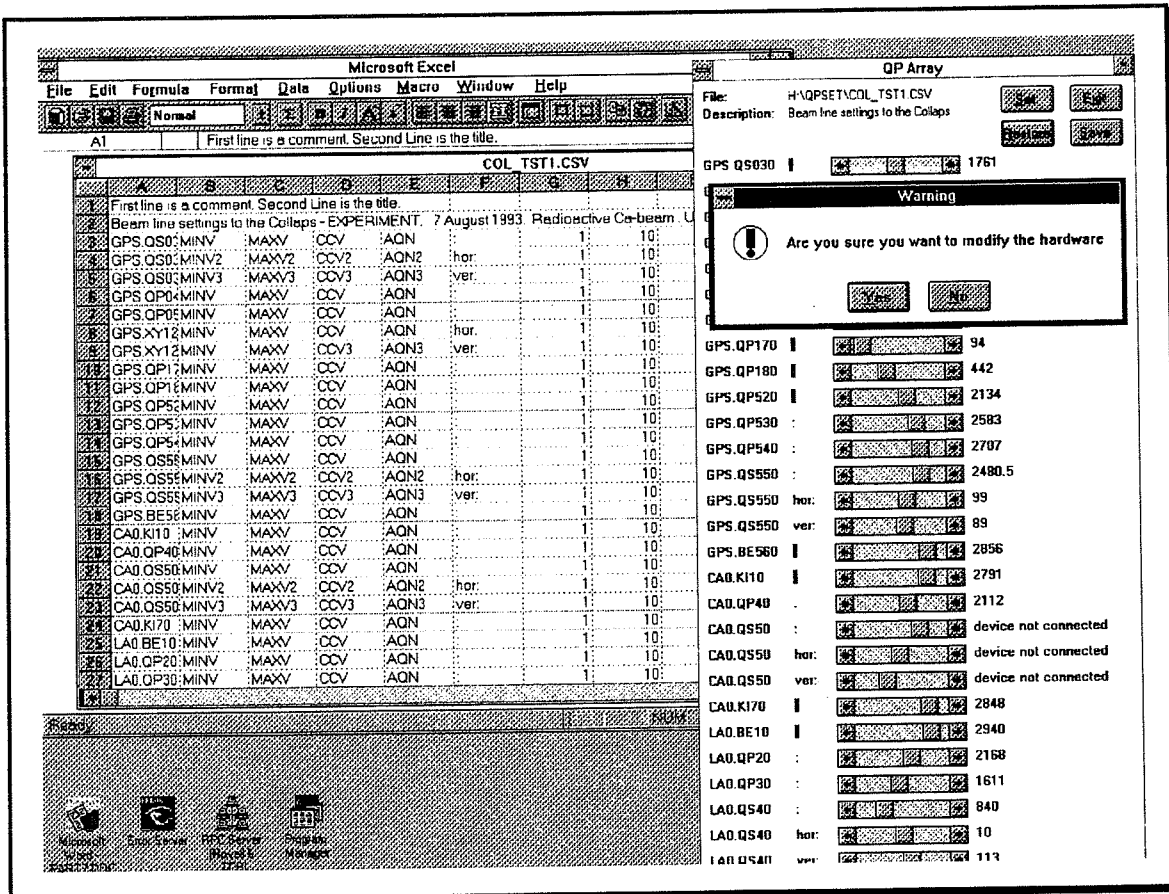


Figure 7.5. The QPArray application and a database file.

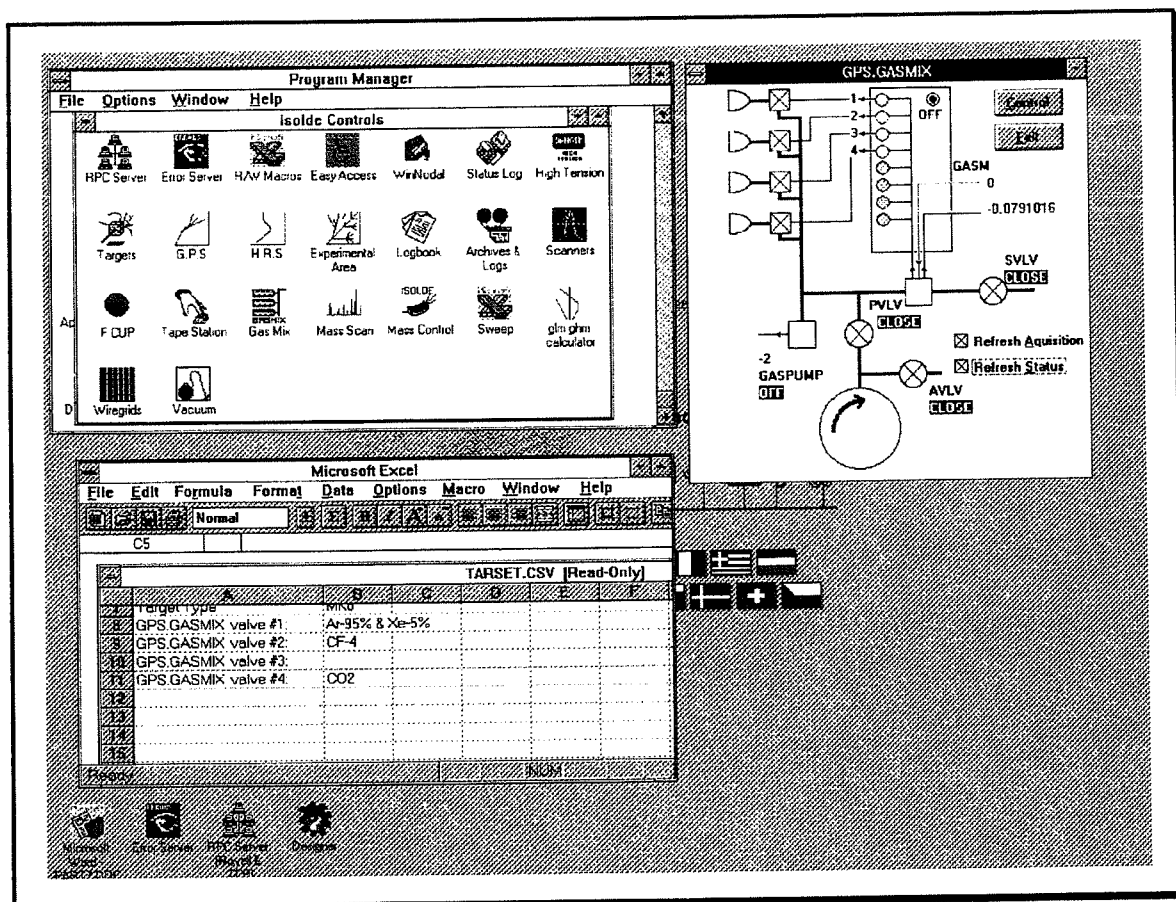


Figure 7.7. The GasMix application and the database file.

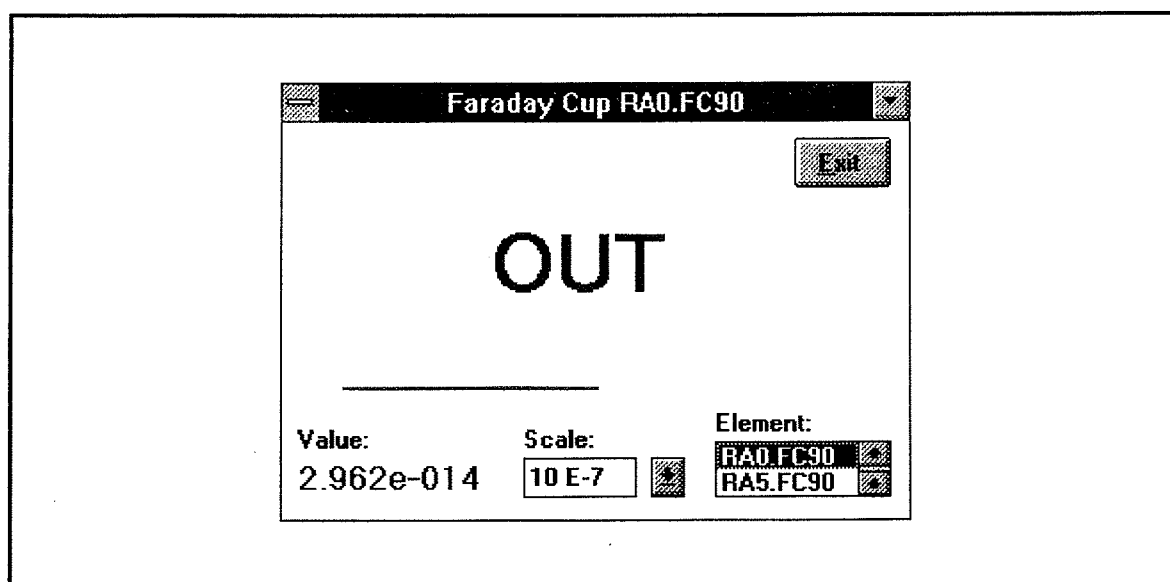


Figure 7.8. Example of the Faraday Cup application.