Computertechnic Laboratory

Connecting a P-NET Fieldbus System to remote Operators via Internet

Technical Description

Writers: Stefan Hutter, Roland Steiner

© ICT, 1999

Internal V1.0 / xxx xxx

ICT, Technical University of Vienna

Abstract

The P-Net Fieldbus is the Fieldbus of the Danish Company Proces Data.The Internet is a technology which get used in the meantime from millions of people all over the world.The most popular programming language of the Internet is JAVA which has been created and optimized for TCP/IP applications.

In this paper is is decribed how this three technologies can bee connected to control a P-Net Fieldbus system over the Internet. It ill be shown that such a Internet-P-NET Fieldbus connection is a good way to enlarge the uses of a P-NET fieldbus system .

 © ICT, 1999

Contents

1 Motivation

The P-Net Fieldbus is the Fieldbus of the Danish Company Proces Data. The communication protocol was published in 1989 and it is now free available over the international P-NET User Organization.

The P-Net Fieldbus is one of three european fieldbus standards, it is mainly designed for process-automatation but it can be used as a universal system too.

The main structure of the P-Net Fieldbus is a physically bus which is united to a ring. With the help of a so called multiport master real networks can be build. A Multiport master is connected to two P-NET fieldbusses and works like a router.With this technology redundant systems can be created.

To controll a P-Net Filedbus with a standard Windows PC, a PCI-card which works as a fielbus-controller and a software called VIGO are required. The software VIGO has got an OLE/dispatch interface so that the full function of VIGO can be used in other programmes or pramming languages like Visual Basic or Visual C++;

The Internet is a technology which get used in the meantime from millions of people all over the world and because of that it is obvious to try to connect the P-Net filedbus system to the Internet. If this gateway exists a big number of new applications for the P-Net would be possible.

The most popular programming language of the Internet is JAVA which has been created and optimized for TCP/IP applications. About that it is a good and comprehensible idea to use JAVA in this case.

An other deciding advantage for using JAVA is that at the user side (client) only a JAVA able Browser is necessary, and in the meantime such programs are installed on the most computers and are available for all operating systems.

That would mean if such a Internet P-NET fieldbus connection is available, a lot of new uses of the P-NET fieldbus would be possible. The entire palette of internet able products (from a PDA to a internet TV) could be used to observe and to control a P-NET sieldbus system.

P-NET fieldbus systems are often used to contorol and observe critcal or important processes, because of that it is necessary to take care of same security requirements. A big number security items are thinkable, from a suitable access control to the use of signed applets and coded client-server connections many itmes could be usefull.

Beside the security requirements an other aspect of the P-NET fieldbus system must be considered, the real time capability. Because of the insufficiently real time capability of the existing Internet it will not be possible to controll or even to regulate a time critical process over the Internet. But it is indeed thinkable that such time critical processes will be definded from the user and then be downloaded to the P-NET fieldbus where they would be executed from a local P-NET controller. The user could define the process, download it, start it and obeserve it over the internet, but he could not

control it in real time.The controll software VIGO would support such functions, it is just a question of expense and money.

So we can see that a internet-P-NET fieldbus connection is a good way to enlarge the uses of a P-NET fieldbus.

2 Problem description

The Fieldbus System and the Internet are both capsulated systems, with no connection between, to manipulate I/O Ports of the Fieldbus you have to make a bridge. There are several ways to solve this problem. We will describe the one used here in the next chapter.

First I want to describe the given conditions. The Fieldbus is connected to a PC via a PC-Card of the P-Net System. This card is needed to configure the System, but you can also change System- and I/O-Settings, therefore I would call this Card a Master. To operate this Card there is a program located on the PC called VIGO. VIGO runs under WIN NT, all functions, f.e. setting up the Fieldbus, downloading files to the System Masters or simply manipulating Ports , are remote controlable with OLE.

OLE is the previous Version of DirectX, you can make Objects and manipulate these Objects from a different Program. This Objects can be created in most programming languages and include all remoteable functions. But we don't need self created programs here on this P-Net PC, we need them on different places outside the plant, in different countries and states.

Additionally there is the problem, that we need this remote controls on different OS, we don't want to store Data on this Systems, no we even don't want to store a Program, you have to install first there, we want quick information about the Fieldbus and want to interact without any installation of special software.

A second reason of holding Data only on one place, the Fieldbus System, is the purpose, that only so there won't be different Versions of the Data and you have less security problems, if you disconnect the interconnection, no unwanted restinformation is on the remote system, which can f.e. belong to a different company.

On the other hand you have the Internet, HTML and Java Applets, all of them are Operating system independent communication protocols or programs. You can find webbrowsers on each OS and most of them understand the above mentioned things.

We want to operate from remote places and from different OS, so internet is a good solution to bring our commands near to the System and get Information from a place nearby our Fieldbus is located.

All the above mentioned inventions are already made, but they can not fix our problem, because these to methods don't work together. We want to control an Fieldbus I/O from an remote Place which is maybe on the other side of the world. All we have is a telephone or datex line, to get or send Information via Internet. It seems to look like a combination of both could fix the problem.

3 Solutions

3.1 Introduction

As described in the last chapter we would have to divide the Problem in different parts to solve it. We did this by making different communication parts. As we begin far away, we took the internet to get nearby our system. but there is no Operating Tool from the P-Net family which can communicate with TCP/IP protocols as the Internet uses them. So the first step brings us maximally to the PC which contains the Fieldbus-PC-Card. On this PC runs VIGO, which controls the Fieldbus and can be controlled via OLE, so we needed only a program which understands OLE and TCP/IP to connect both. I will describe this and many more of our ideas next.

3.2 The way to define a solution

As I wrote in chapter 2, we wanted our product to run on different OS, so there is one programming language, which can be compiled to a runable file, which can be executed on most OSs, it's JAVA, the hint with JAVA is, that the Sourcecode is compiled and/or linked in two stages, the first one is System independent, the so created File needs a virtual machine to be executed. But it can't be modified and viewed, as you can do this with a Sourcecode.

A special form of these applications is called applet, these files are executed with virtual machines located in webbrowsers, the Frontend is the webbrowsers Frontend, you can see the programs (applets) surface in the browser window. Most browsers have a virtual machine with it, or you can load some plug-ins. Browsers are delivered with OS, so you won't find a machine with an OS, which allows TCP/IP connections with no browser.

The second advantage of JAVA is, that the applets are stored on the server, you most load them each Time you use them new. They only work, if you are connected to the server. The last to sentences are not really exactly true, because you can store JAVA-Applets on your local system as you can store HTML pages local. But the Applets we used needed a connection to work correctly, because they built up a socket communication[1] to communicate with the server, this is only possible, if you are online. So there was no way to work correctly offline. All I wanted to say in this paragraph is, that there is no data to save on the remote system, what's good for security habits.

As you can see JAVA is a very fine thing to get programs and Data to remote systems with no installing of programs and having data inconsitence and security problems. But is it good for the other side of the interconnection, too ?

[1] see below in text for more information

On the other side of the communication line, we don't have security problems and it is possible to install software on the PC, because this machines special examine is to control the Fieldbus. There are some Points for and some against JAVA on the Server based side:

1. JAVA is not very fast, because you have to compile the only precompiled code before executing.
2. but JAVA on both sides allows to communicate with objects, types which understands only JAVA. As I mentioned, we used a socket connection, this way of connection is implemented in most programming languages, so we could send types which are part of a language between programs of different languages. objects have the big advantage, that you don't communicate only with data, but also methods to manipulate them.
3. A big problem is, that JAVA is a programming language which is object oriented and is able to handle with common objects in different programs or even on different machines. This is a benefit and not a problem you would say, but it is a problem for us that OLE or DirectX is not implemented in Java, because these two methods would do the same as you can do with JAVA Objects. You can handle OLE Objects but it is compared to other languages as C++ or Visual Basic very difficult, you need special techniques called JAVA Beans[1]. The reason is, that OLE is a Microsoft product, JAVA is a SUN product and works on OSs of other producers to0.

After all we tried to solve the example with a minimum of different programming languages, because it's easier to understand and to modify the product if you do it so.

What we have now is a JAVA Application running on the P-Net PC, it's a server program which communicates with it's JAVA Client, which you can find on the same PC. This is a simple Client-Server solution, the speciability of our product is, that we use objects to communicate. Our simple Client-Server Application uses a socket communication system, which depends on TCP/IP, with TCP/IP communication you can address PCs all over the world, they only have to be connected to the Internet. So we have here already a worldwide Client-Server connection. What we want is, that the Clientprogram is only located on the remote PC during the execution. We made the Client an applet, which is located on the Server-PC. If you want to communicate with the Server, the foreign PC loads the Apllet onto his workspace (via Internet) and executes it with the browser of his platform. How this exactly works is described later.

What we now still need is the OLE interaction of VIGO with our JAVA Server, this is described in Chapter 5, but at this time we didn't choose this solution, because of the complexity of interacting OLE with JAVA Beans. We created a second Server an C++ Server who interacts with VIGO an the JAVA Server.

We used C++ because it's like JAVA object oriented and a complex high level programming language, you don't have the possibilities of applets or other stuff de-

[1] We will explain how to work with JAVA Beans in Chapter 5.

scribed before for Internet communication, that was the reason, why we didn't choose it for the whole project. The goal should be for further works, eliminate all the C++ stuff.

Between both Servers we needed a second TCP/IP connection, but now we had a local Host, which means that we don't need Internet connection, because both programs were running on the same machine, even in the same subdirectory.

More Information about how to work with the product is found in a chapter of it's own.

3.3 Conclusion

We have to PCs, standing on different places. The one coupled with the Fieldbus is running under WIN NT, a program called VIGO which interacts with the P-Net is running on this machine, VIGO is OLE compatible, we used a C++ Server, we wrote ourselves, to create an OLE Object of VIGO and manipulate the P-Net out of this local Server.

A second Server, a JAVA Application, which is also installed on the local machine and built in this project, communicates with the first Server with a TCP/IP socket communication. On the other hand this Server has a second TCP/IP communication with a remote Client. This Client is a JAVA applet, which is stored on the local machine, but it can be loaded to the remote PC before execution.

The final goal is that the remote PCs browser on which the JAVA applet runs communicates with VIGO. You can listen and interact things on the Fieldbus out of your browser anywhere.

4 The P-NET Fieldbus

The P-Net fieldbus is a fieldbus of the Danish CompanyProces Data. The communication protocol was published in 1989 and it is now free available over the international P-NET User Organizartion.

The P-Net fieldbus is one of three european fieldbus standards (beside the danish P-NET there is the german Profibus and the french FIP Bus), it is mainly designed for process-automatation but it can also be used as a universal system. The main structure of the P-Net Fieldbus is a physically bus which is united to a ring. With the help of a so called Multiport master real networks can be build. A Multiport master is connected to two P-NET fieldbusses and works like a router.With this technology redundant systems can be created.

Up to 125 nodes can be connected on a single bus, this nodes can be master or slave.A slave is only allowed to use the bus for sending back the reply data after a master has questioned the slave. The bus access is defined through the virtual-token-system, because of that the P-NET is a system where real-time access is guaranteed.

A master, which do not have the token at the moment, can be used from the master which has the token at the moment, as a slave.

Controllers are the most intelligent devices in a P-NET system, they can be programmed with a high-level programming langue called ProcesPascal. The developing of a ProcesPascal programme happens naturally on a normal Windows PC.

But also the less intelligent nodes of a P-NET system, like a I/O units, are smarter then such units in other fieldbus systems.This less intelligent P-Net units can be programmed with a simplified assembler language called CalculaterAssembler and they are able to do some simple tasks on their own like SI-unit conversion's or simple PID regulations.

Every P-NET unit includes multiple channels. A channel is a group of value-objects like input- and output values, minimum and maximum values.For the most important channles standards have been defined. Because of that similar channels can be found in P-NET units from different manufacturers.

The software VIGO, which is developed for the P-NET, creates objects for Windows which represents the units of the P-NET. VIGO has got an OLE interface and because of that every windows programme, which can handel such an OLE interface, is able to control the P-NET system.

An other opportunity for constituting the P-NET process is to use the PD5020 controller. This controller has got connection's for a monitor,a computer mouse and a keyboard and the software GCS enables you to create a fine visual representation of your P-NET process.

After that we would like to explane some important principles of the P-NET Fieldbus:

The **bus-access method**, the **routing method**, the **channel structure** and the **variables-spelling method**.

4.1 The bus-access method: Virtual Token Passing

The P-NET fieldbus is a multimaster able system. Because of that the access to the bus must be defined, every unit on the bus have to know exactly when it is allowed to send data and when not. This task is realised in the ISO/OSI layer two, the method is called Virtual Token Passing.

Every one of the maximal 32 master units which can be connected to a single P-NET bus has got a Node Adress, starting at one up to 32, and a Idle Bus Bit Counter. This counter counts the idle bus periodes. Further more every master has got an Acces Counter which get incremented when the Idle Bus Counter reaches the numbers ...,40, 50, 60,... etc.If the value of the Access Counter is the same as the value of the Node Adress, this master has got the token and it can use the bus for sending data. If the value of the Access Counter is bigger then the number of masters of the bus the Acces Counter will be set back to one. Because of that it is necessary to store the number of masteres on the bus in every single master unit.

4.2 The routing method

In the definition of the P-NET Fieldbus system a very simple but effective routing method is implemented.

For example, if a master has a request to a client the master has to indicate the exactly way to the slave, with all the multiport masters which could be between the master and the slave. At the end of the adress statement the master writes the simple adress of the client and his own adress.

This adress statement can be used from the client in a very simple way to send its replay back to the master.

The definitions of the routing statements (they represent the construction of the P-NET system) have to be known at the programming time, that means: If the structure of the P-NET system changes, the programme of the system has to be modified too. But if you do the programming in a smart way, you must not change very much. It should be sufficient to change the definition part of the programme. A P-NET fieldbus can be a redundant systems, that means that there could be more then one way the data could flow between a master and a slave. The porgrammer has to choose the way the data should take. That means that the P-NET system is not so intelligent to choose the way the data should go. So if there is a problem at the bus, for example one part of the system is broken, the system will not find a way by itself to avoid the damaged part, even if it would be possible because of a redundat system structure.

4.3 The channel structure

A simple P-NET unit, for example an analog input, has the assignment to catch data and send it to an other unit. But to process this data more information is necessary. Information about how to scale, to transform, to filter and so on . The summation of this variables and functions in form of an object is called a channel. Because of that ist is recommended to define an object orientated interface modul where more then one channel can be put together if it is necessary. Therefore it is possible for a programme to have access to a unknown P-NET unit as long as this unit uses well known channel types.

A channel is subdivisioned in 16 registers, each one has its own logical adress, the so called Softwire Number (SWNo). This 16 variables or constantes can be different types or records of different types,this depends on the requirements of the channel. Also different memory types can be used (ROM,.RAM,EEPROM,...).

In general channels can not only be defined for measurement- and regulation-tasks, they can be used as well for data transmission tasks, for example a printer.

4.4 The variables-spelling method

To have access to any variables in the P-NET system the physicaly place of the variable has to be known. The affiliated and necessary informations are: the before defined Interface Module, the P-NET adress and the number of the P-NET port.

The structure of the variable adressing looks like:

```
Nameofthefunction:NameoftheModule AT NET:(P-NET Nr.,P-NET ad-
ress);
```

for example:

```
Waage:PD3230 AT NET:(1,$30);
```

Further more there is the possibility to define a variable indirectly. That means that a shorter or more significant name can be assigned to a variable without allocating a new memory block. Because of this assigment the new variables gets the type of the originally variable.

4.5 Simple Exampel: RoST 1

We wrote a sipmle program for the controller PD4000 to show the principle structure of a Proces Pascal program.

The structure of a ProcesPasacal program:

```
PROGRAM ROST1 [ objecttype=4000; capabilities= nobitaddress];
/* include files
(*$I'..'*)
```

```
(*$I'..*)

/* global data
CONST
....
TYPE
...
VAR
...

/* Task Definition
TASK YourFirstTask;
CONST
...
TYPE
...
VAR
...
BEGIN
/* Programmanweisungen
END; // Task

END. // main Programm
```

In a ProcesPascal programme more then one tast can be processed at the same time.

Our programme RoSt1 shows the temperature and the weight of the ICT P-NET system on the PD4000 display. Press the first key to toggle between light 1 and light 2 , press the scound key to switch the lights on or off.

For the listing of RoSt 1 look at the Appendix.

4.6 Still simple example: RoSt 2

With our second example we wanted to realise the following functions:

- the full controll over all ventils/lights/pumps/mixer/heater of the ICT P-NET
- ? to display all sensors of the ICT P-NET
- ? to realise a small process with the ICT P-NET

Out of that we created three differnet ways to operate with the PD4000:

screen1: displays some infos about RoSt 2 and shows the function keys to change the screen

screen 2: a small process can be started and stoped, the current process data is shown at the display

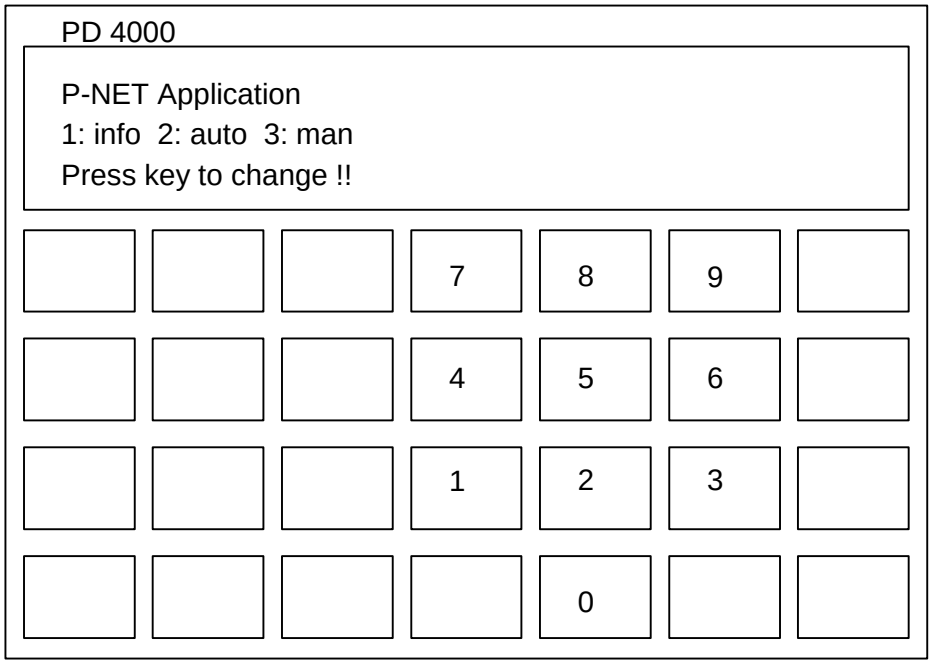
screen 3: all sensores are shown at the screen, use the keys to controll the ICT P-NET

Don't forget: Be carefull with the heater, the hole ICT P-NET can burn down if you forgett to switch it off !!

The Listing of RoSt2 is show in the Appendix.

4.6.1 screen 1: info

At screen 1 the following text will be shown at the PD4000 diaplay:



4.6.2 screen 2 : auto

Following process can be started at screen 2:

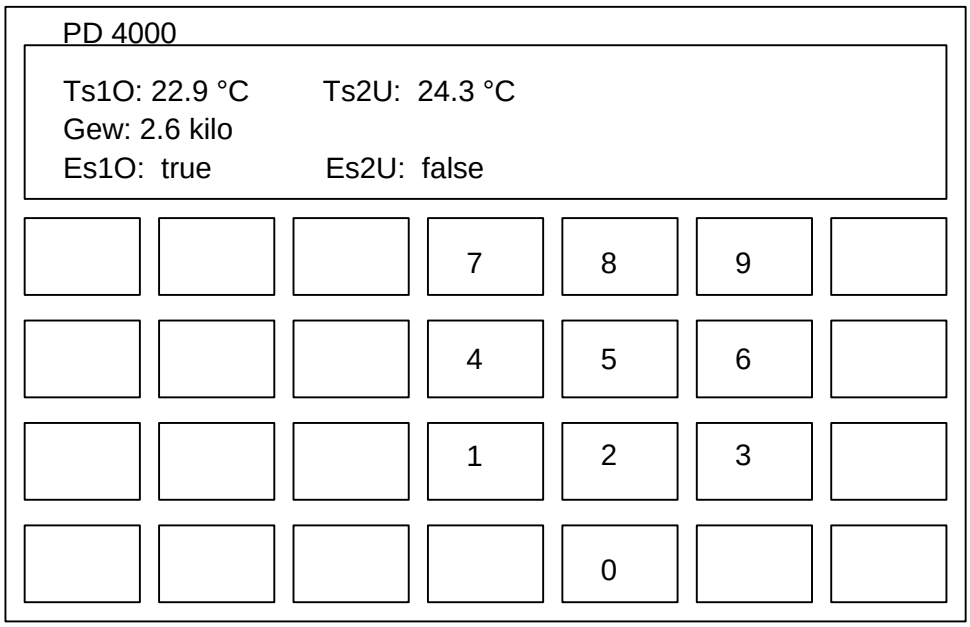
- Press key to start the process
- Fill tank 1 until end sensor 1, switch on light 1
- Increase the temperature of tank 1 up to 3°C,switch on light 2
- Unfill tank 2
- Fill tank 2 with 1.4 kilo from tank 1, display the weight, switch on light 3
- display message <program finished>, stop process

While the process is running the current process data is shown at the display of the PD4000.

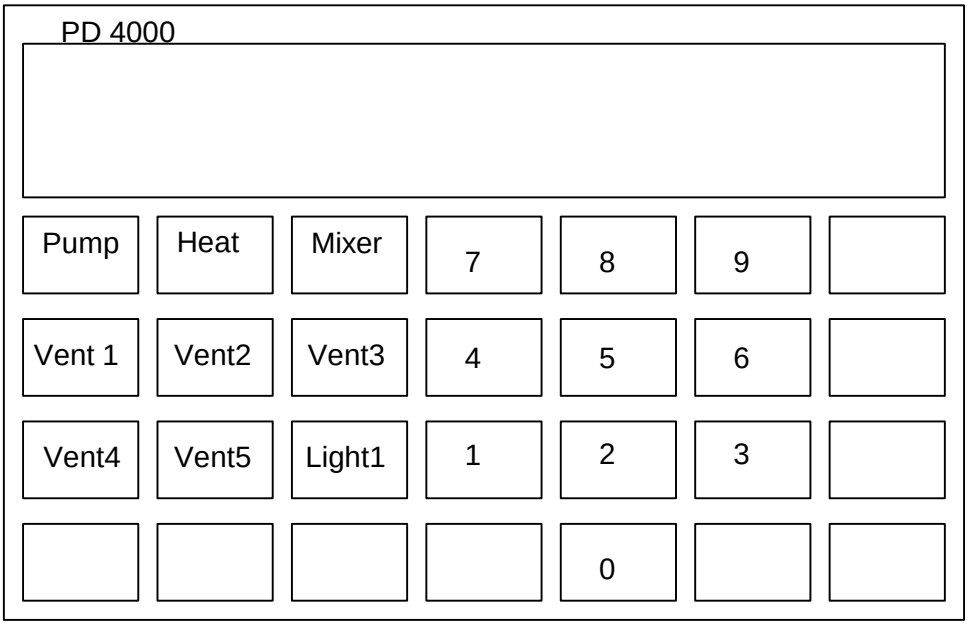
4.6.3 Screen 3: man

At screen 3 the sensores are shown at the screen and the P-NET can be controlled with the function keys.

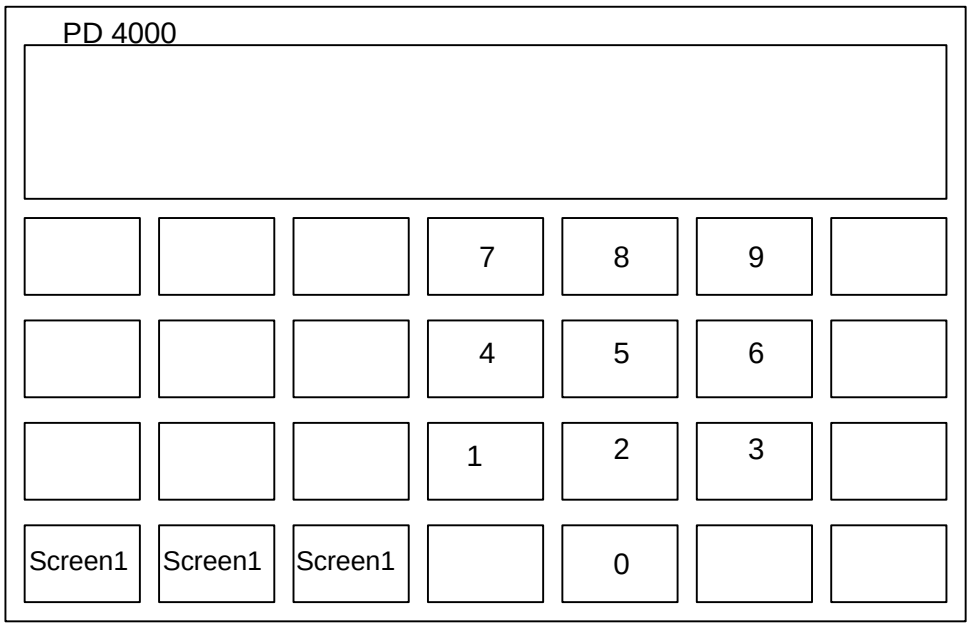
> The following text will be shown at the display:



> With the following keys the P-NET can be controlled:



> To toggle between the screens use the following keys:



5 The VIGO OLE controll

The main ponit of the server programme is beside the client server connection, the control of VIGO over the OLE/DISPATCH interface.So, first of all, let me describe this OLE interface in a few words.

5.1 The OLE interface

OLE (Object Linking and Embedding) is a technologie for exchanging data between different Windows programmes which was developed from Microsoft. In opposition to DDE, which is desingned for exchanging small quantity of data, OLE is based on objects.Objects can be for example complete programs, parts of programs, charts ore a simple value.OLE2, which is a part of windows since Windows 3.11, enables windows programmes to create so called OLE-automation-objects. This objects can be used and be included from other windows programs. Programmes which can export such objects are called OLE server.That means that the applications which make use of the objects (the so called clients) can use the properties and the methods of these objects.

Because of the Internet boom of the last years, Microsoft modifed the OLE interface and createed a new name for it: ActiveX. With the ActiveX technologie ist is possible to use objects from applications running on differnet computers all over the Internet.This ActiceX technologie is a competitor of the JAVA technologie, but ActiveX is just designed for the Windows world. One of the great features of JAVA is that it can be used for all operating systems, from a high-end workstation to a washing machine, so i realy don't know if it is a real competition between Java and ActiveX. But i know that Microsoft sells JAVA too in the meantime, so JAVA won the battle, din't it?

So when you try to use the OLE interface with Visul C++, for example, you just have to do the following:

```
vobj= new IvigoPro;
vobj->CreateDispatch("Vigo.Pro");
vobj->SetPhysId("Waage.Digital_IO_1.FlagReg[7]");
```

The thing we wanted to try was to use the OLE interface with JAVA. Because our task was to controll the P-NET over the Internet, and for the front end of our programme we had to use JAVA so we thought it would be fine to use just one programming language in the hole project.
After reading the JAVA dokumentation and some JAVA books and some JAVA scripts it was clear that it would not be so easy as we thought. The problem is that JAVA is not designed for the WINDOWS platform, it is designed for all computer platforms. But OLE is a special WINDOWS interface and the JAVA core can not handle with this interface.
So, what should we do? Nowadays every little child knows that the hole knowlege of mankind an much more is availible over the Internet. And so we searched with diffe-

rent search engines and different terms, and we found a lot. But all what we found was about OLE/ActiceX *or* about JAVA, or about: *what is better*, JAVA or ActiveX? But nothing about how to connect JAVA and ActiveX.
Of course Java has got a concept for including objects from foreign applications, this concept is called JAVA BEANS. But in the BEANS documentation they are always talking about invoking JAVA objects in JAVA programmes (or C++ objects in JAVA programs), so it was not so clear how wide this concept is.
Finaly, we found a solution. The missing stone in the wall was the word „Bridge to ActiveX".
With such a „Bride to ActiceX" it is possible to connect OLE with JAVA with the help of the JAVA BEANS. So, i think this is the right place for a little excerpt out of the JAVA BEANS Tutorial from SUN:

5.2 Java Beans Concept

The JavaBeans API makes it possible to write component software in Java. Components are self-contained, reusable software units that can be visually composed into applets or applications using visual application builder tools. JavaBeans is a core JDK1.1 capability: Any JDK1.1-compliant browser or tool implicitly supports JavaBeans. JavaBean components are called Beans. A "Beans aware" builder tool maintains Beans in a palette or toolbox. You can select a particular Bean from the toolbox, drop it into a form, modify it's appearance and behavior, define its interaction with other Beans, and compose it and other Beans into an applet, application, or new Bean. All this can be done without writing a line of code.

The following list briefly describes key Bean concepts:

Builder tools discover a Bean's properties, methods, and events by introspection. Beans support introspection in two ways:

> By adhering to specific naming conventions, known as design patterns, when aming Bean features. Bean introspection relies on the core reflection API to discover Bean features via design patterns.

> By explicity providing property, method, and event information with a related Bean Information class. A Bean information class implements the BeanInfo interface.

Properties are a Bean's appearance and behavior attributes that can be changed at design time. Properties are exposed to builder tools by design patterns or a BeanInfo class.

Beans expose properties so that they can be customized at design time. Customization is supported in two ways:

By using property editors, or by using more sophisticated Bean customizers. See Chapter 9 of the JavaBeans API Specification for a customization discussion. Beans use events to communicate with other Beans. A Bean that wants to receive events (a listener Bean) registers its interest with the Bean that fires the event (a source Be-

an). Builder tools can examine a Bean and determine which events that Bean can fire (send) and which it can handle (receive).

Persistence enables Beans to save their state, and restore that state later. JavaBeans uses Java Object Serialization to support persistence.

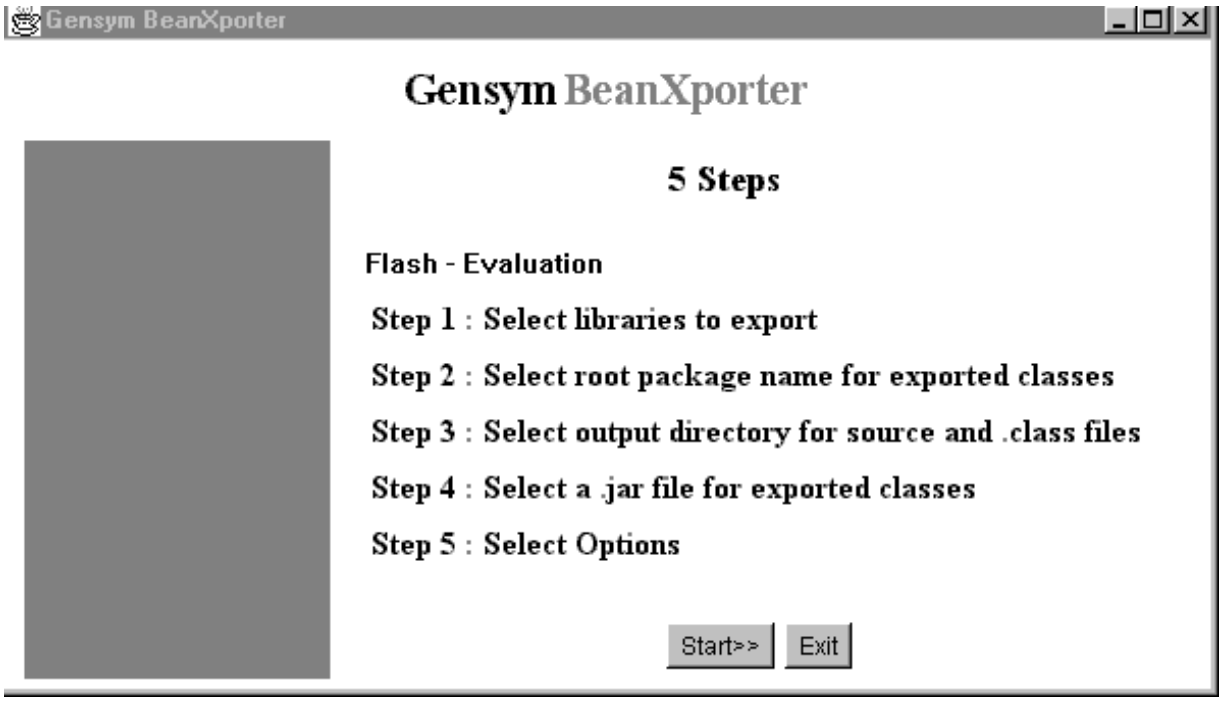
A Bean's methods are no different than Java methods, and can be called from other Beans or a scripting environment. By default a all public methods are exported. Beans can be used with both builder tools, or manually manipulated by text tools through programmatic interfaces. All key APIs, including support for events, properties, and persistence, have been designed to be easily read and understood by human programmers as well as by builder tools.

To handle with such BEANS you need the BEANS DEVELOPMENT KIT (BDK). It is free availible form SUN over the Internet.

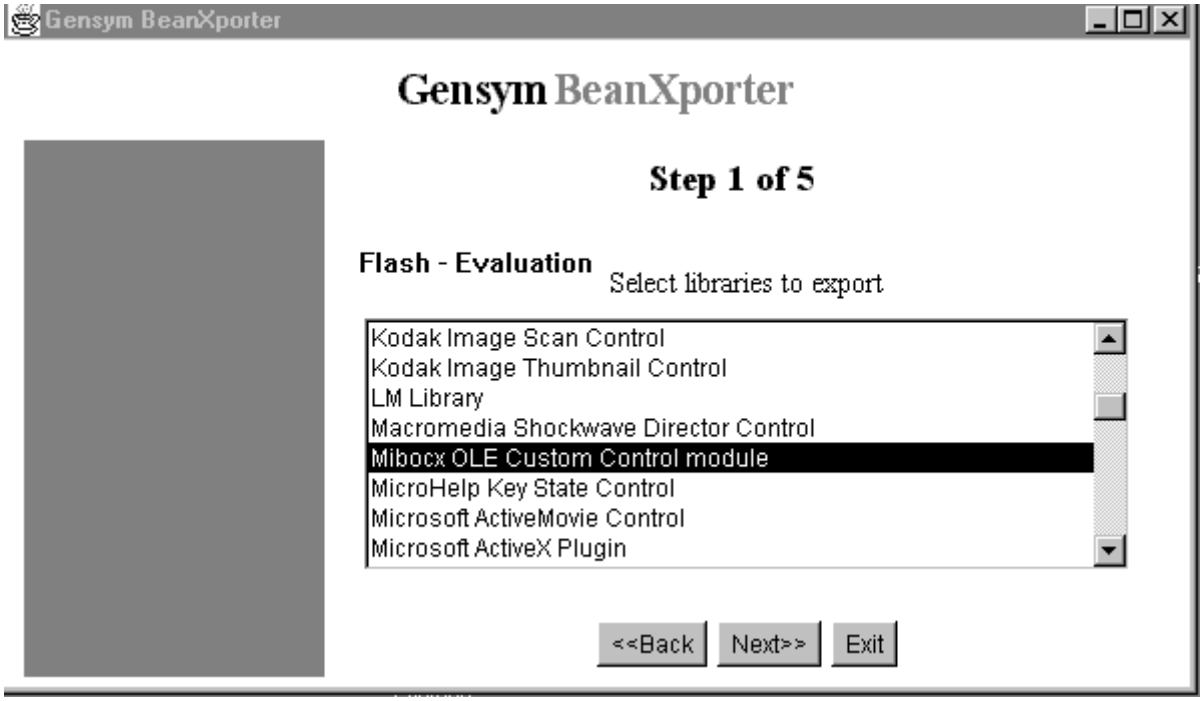
The „Bridge to ActiveX“ is a JAVA BEAN which can be created with a separate tool, for example with the tool BeanXporter from Gensym.

To create such a BridgetoActiveX-JavaBean you have to install VIGO on your Computer, a JAVA Developmet Kit, a JAVA Runtime Enviroment, the JAVA BEAN Development Kit and the „BridetoActiveX“ tool, for example the BeanXporter (you have to take care about the Java version and the CLASSPATH variable). After that it is simple to create the Bridge:

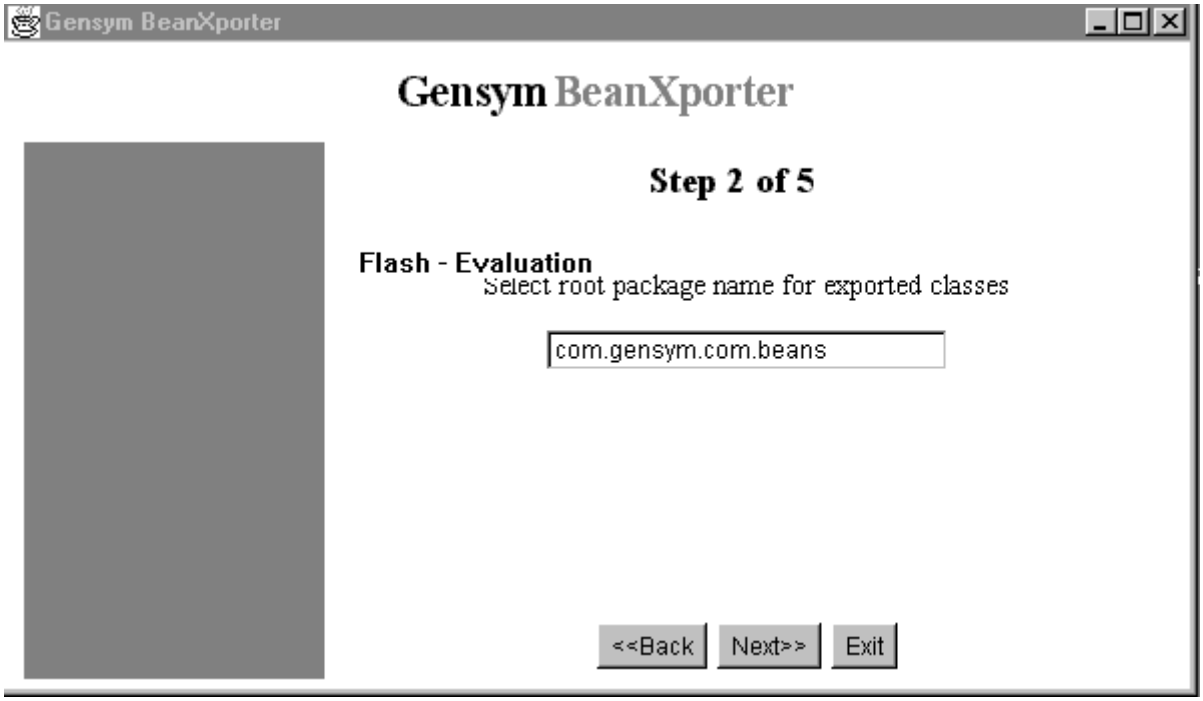
Start the BeanXporter and make 5 steps



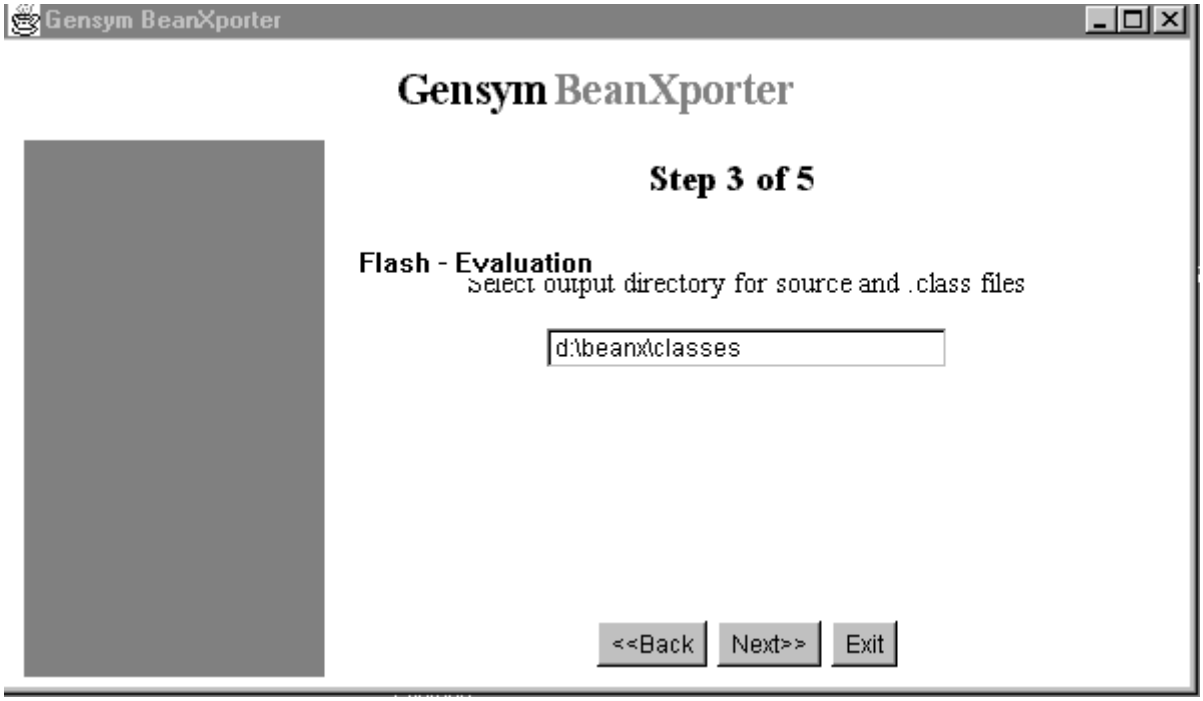
Picture 5.1: Create a OLE Bean (Step1)



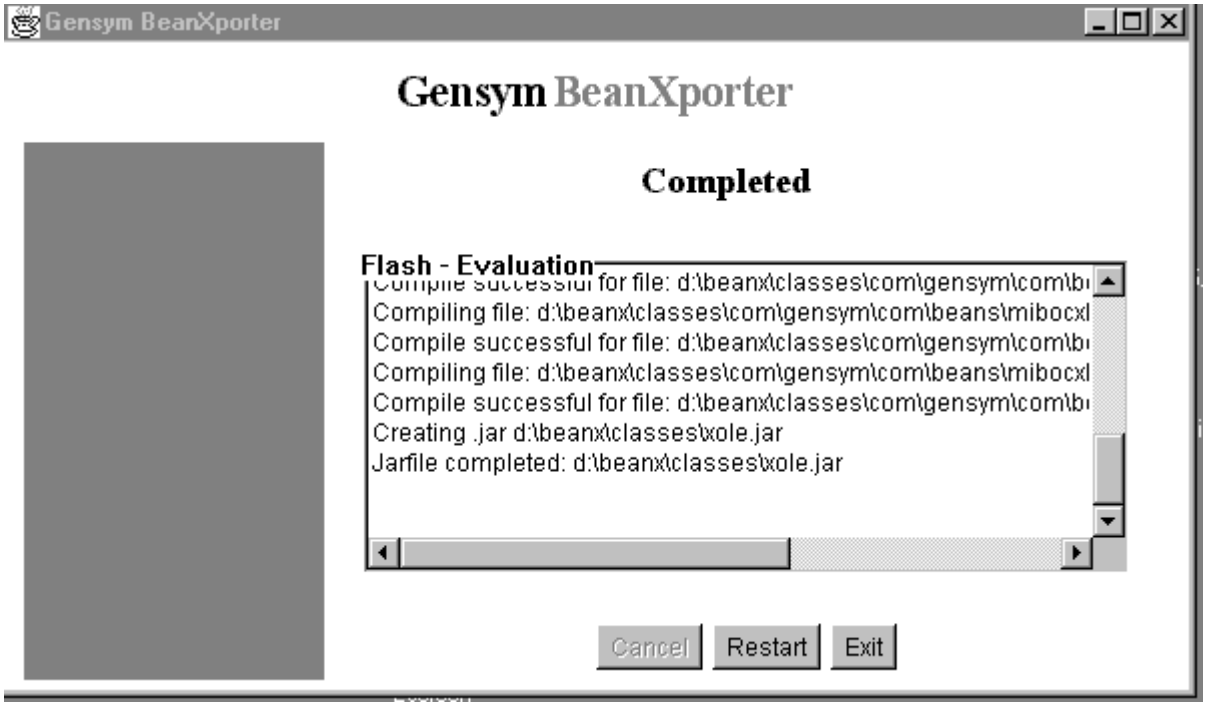
Picture 5.2: Create a OLE Bean (Step2)



Picture 5.3: Create a OLE Bean (Step3)

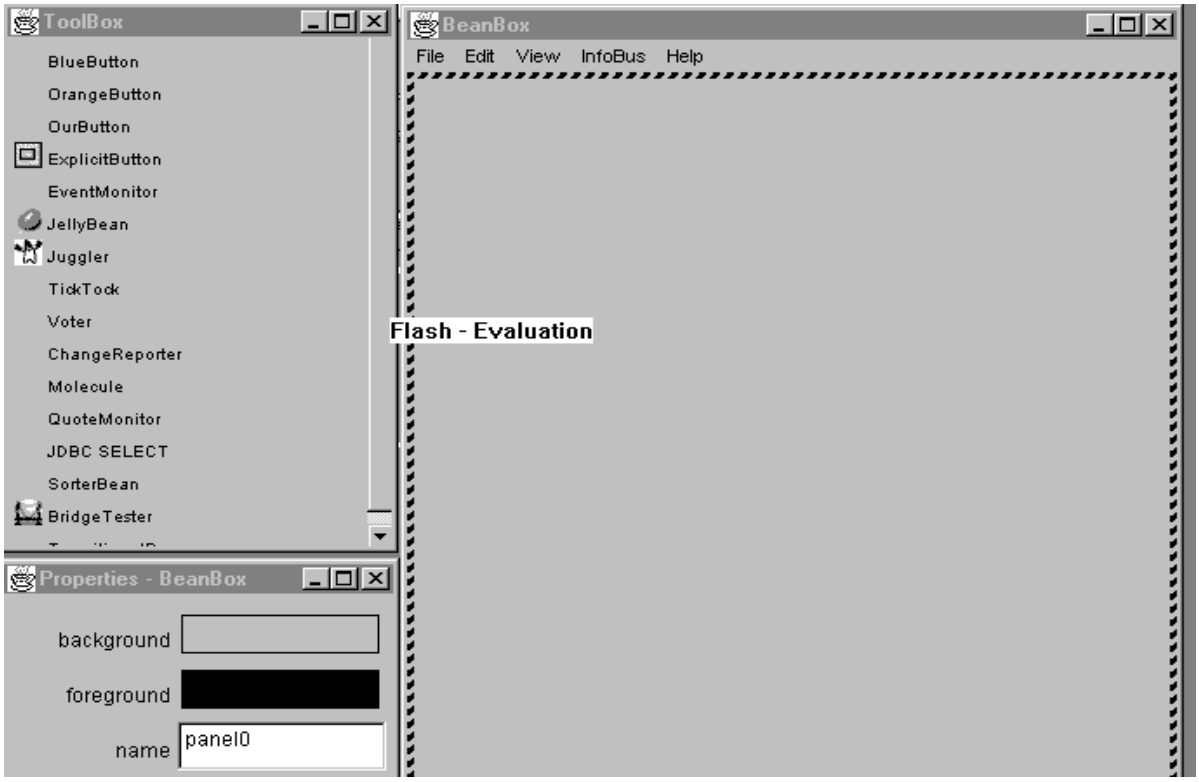


Picture 5.4: Create a OLE Bean (Step4)

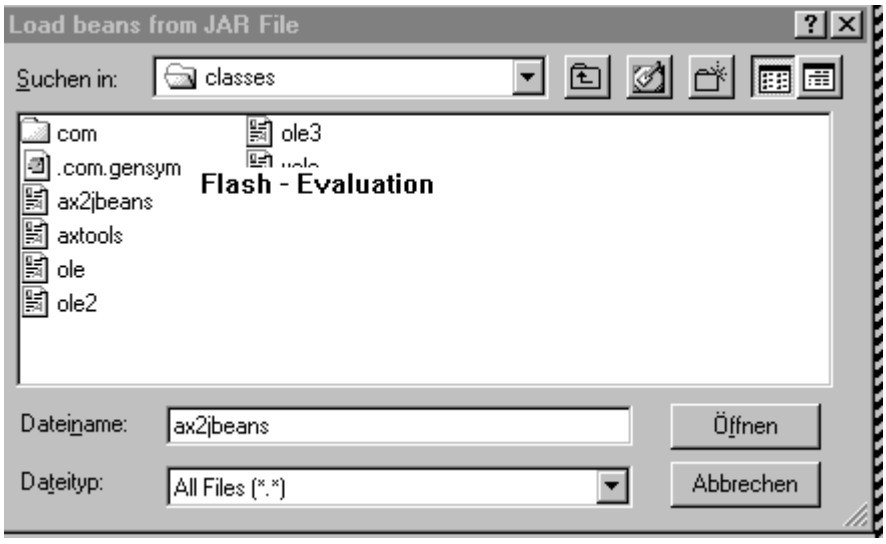


Picture 5.5: Create a OLE Bean (Step5)

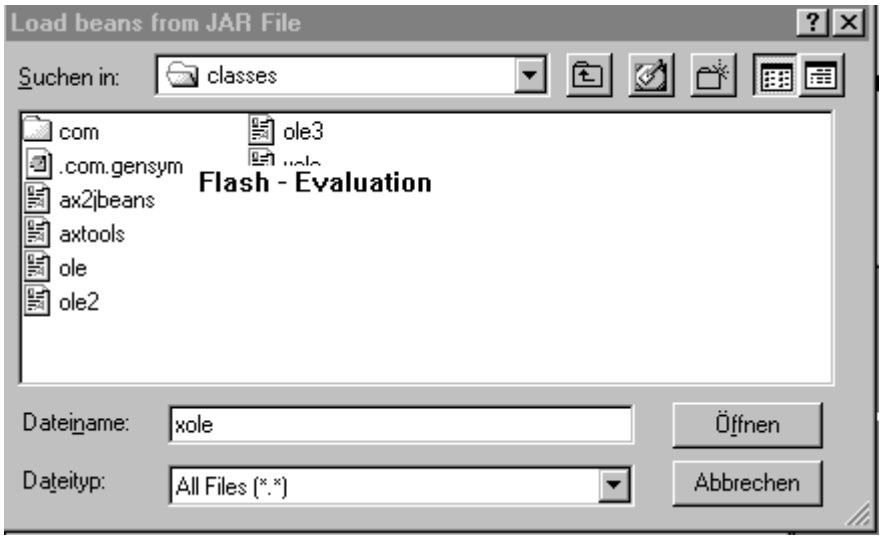
Start the Beanbox out of the BDK and insert your Bridge (dont forget to include the BeanXporter Tools first)



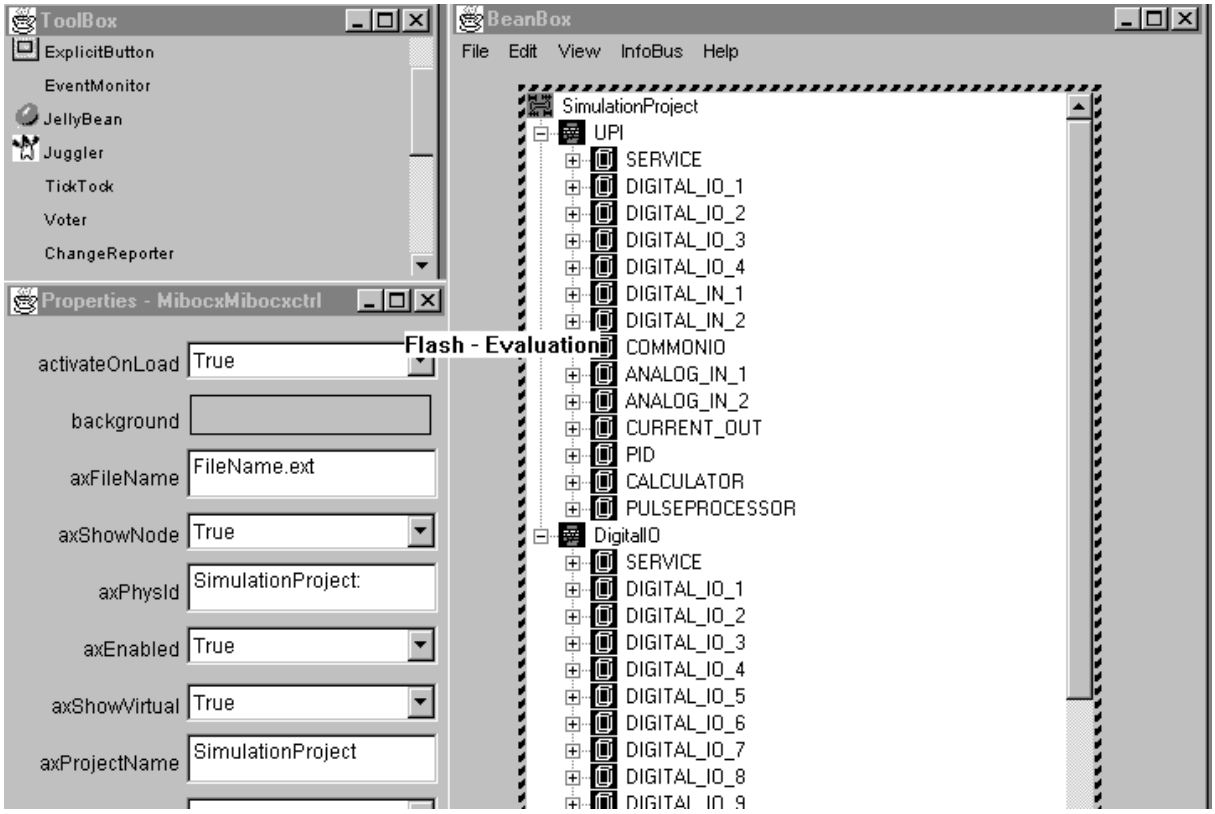
Picure 5.6: Insert the Bean into the Beanbo



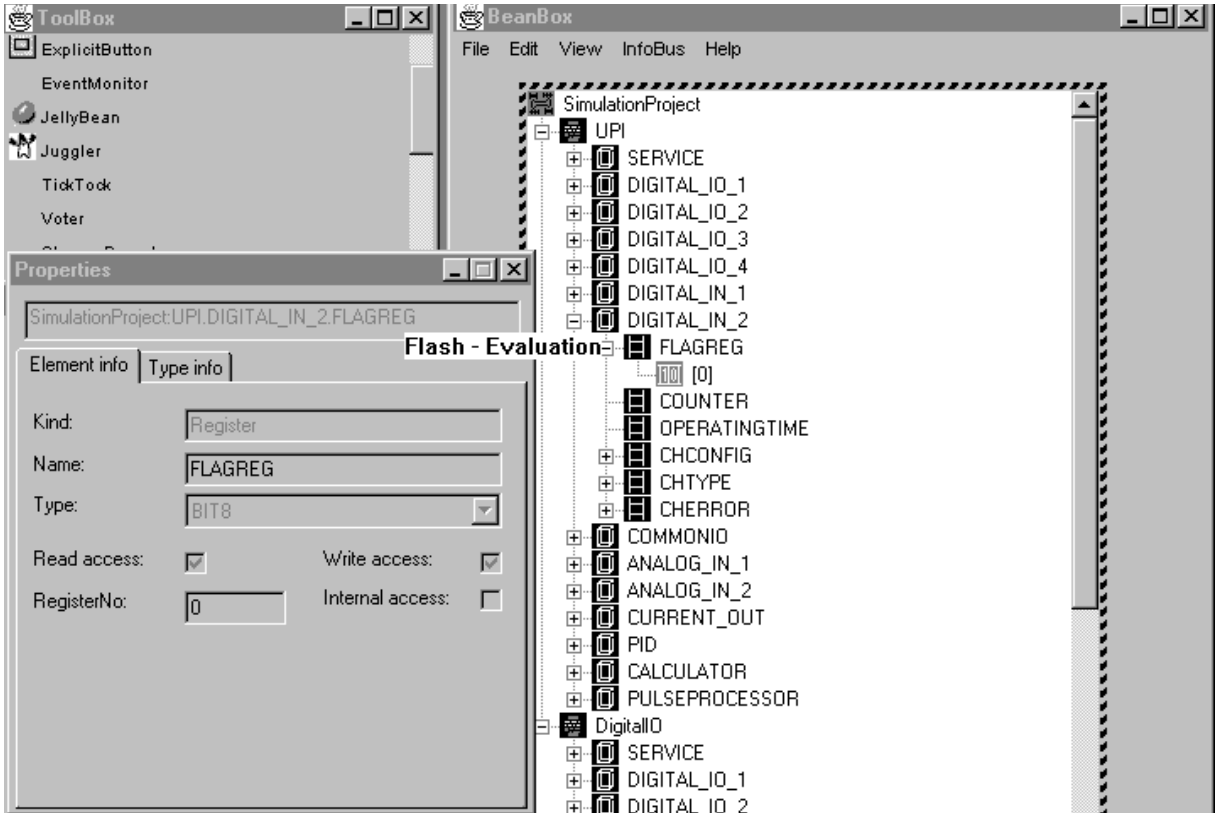
Picure 5.7: Insert the BeanXporter Tool Bean into the BeanBox



Picure 5.8: Insert the VIGO OLE Bean into the BeanBox



Pictur 5.9: The VIGO OLE Bean is included in the BeanBox



Picture 5.1: The VIGO properties can be edited

And now we are ready. We have created the Bride and included it to the BeanBox, and the result is great. The hole power of VIGO is included in a JAVA enviroment, you can see and controll the hole project-tree of the P-Net, you can display end edit every single property of the VIGO object. And that seems to be the problem. We have more functions as we want, the BridgeBean is very big and we need a powerfull JAVA development enviroment to insert this bean in our JAVA server programme. We tried it with JdesignerPro3.0 form BulletProof and JAVA CAFE from Sympatec, but it did not work. The compiler was always displeased and after some tries we gave up.

I think it could work well when sombbody would invest more time in it, but when you know how easy you can handle the OLE interface with VBASIC or VC++ it is thinkable that there is an easier solution.

We felt the icy breath of time in our neck and so we decided to realise the OLE-server part in Visual C++.

6 Internet Connection

6.1 Introduction

This chapter will tell you how to make a simple JAVA connection as we use it in our project twice and how we defined our Frontend, which is a compilation of several applets. We will talk about binding applets into HTML Code, communication between applets, designing a Frontend Layout and other JAVA stuff we needed.

6.2 Java Socket Connection

We have two different communications in our project, the remote one has a JAVA Server and a JAVA Client, the local one has only a JAVA Client, the Server is wrote in C++. A Socket Connection is not only a one-to-one connection, you can have several Clients interacting with one Server. Therefore you most work with Threads.

Threads are pieces of Sourcecode or even whole programs, which can be started for more then one Time and executed simultaneous. A problem could be, that they need the same Data at the same Time and you have to look, that they don't want to read or write to memory at the same Time (mutual exclusion).

We used with this technique not only two (Client and Server) programs, but we wrote three, the third one was a ServerThread which is started at the Time a Client opens a socket connection. At first I want to explain the ServerMain program, which is started first:

Java.net is a library which includes all TCP/IP, socket, port and other Client-Server commands:

```
import java.net.* ;
```

The type Socket and ServerSocket are also a part of java.net (the Server needs a ServerSocket, the client a Socket, ServerSocket is a special type of Socket), each socket has different ports for several different accesses. We don't won't different ones, cause all Clients want to have the same data. "ServerCon" is the name of the thread, "doit" is an object of type "ServerCon" :

```
ServerSocket server;
Socket s = null;
ServerCon doIt;
int port = 4711;
```

The try function is used for operations which may end in errors, all of them are cached at the end of the section, most times you print them to the system and exit the program.

After you created a new ServerSocket you wait for a interaction of the Client with .accept(), if the Client is online you create a new thread of “ServerCon” for this special client:

```
try {
  server = new ServerSocket (port);
  System.out.println("Server running....");
  while(true) {
    s = server.accept();
    System.out.println("Client online");
    doIt = new ServerCon (s);
  }
} catch (Exception e) {
    System.out.println("ServerMain " +e);
    System.exit(1);
}
```

This is the whole “ServerMain” program, next I want to explain the thread. Here you can find the java.net library ,too. There are two different classes, java.io for reading and writing objects and strings and java.util. The big different between a thread and a normal application is shown next:

```
public class ServerCon implements Runnable
  private Thread t;
  public ServerCon () {
    t = new Thread(this);
    t.start();
  }
  public void run() {
  ...
  }
}
```

First the class has to be declared as runnable, you have to define an object of type Thread, which creates the public class of it’s self, “this” means, that you create a new thread of this class. After creating it, you must start is. All code which is standing in run() is executed after starting.

Normally you only have to make readers and writers to interact with the client, but this thread is a client for the C++ Server, too. So we must create a socket, to communicate with the second server, too:

```
socket_c = new Socket ("",4712);
```

Now we can really start with building a communication, we have four, a reading and a writing end from the applet and a reading and a writing end to the second server, between this two ends we have to reformat the messages:

```
ObjectInputStream inStream = new Objectinputream
```

```
(socket.getInputStream());
ObjectOutputStream outStream2  =   new  ObjectOutput-
Stream(socket.getOutputStream());

Printwriter c_out = new PrintWriter(new BufferedWriter(new Out-
putStreamWriter(socket_c.getOutputStream() ) ) );

BufferdReader in = new BufferedReader(new InputStreamReader
(socket_c.getInputStream() ) );
```

As you see, the four here used variables are declared in the definition part, this is allowed. All of them have socket.getInput(or Output)Stream, this is the elementary instruction to get a stream, afterwards a new object (ObjectOutput, InputStream-Reader, ...) is built to give the data transmitted a certain format. For Strings a normal Input- or OutputStreamReader is essential, to use the print and println commands you have to set a Printwriter onto an OutputStreamWriter. For Objects you need ObjetOutput- and –InputStreams.

Next I want to explain how to read and write, but before I have to explain the Object we use here:

```
class protocoll2 implements Serializable{
  public String command="";
  public String unit="";
  public int data1=0;
  public int data2=0;
}
```

This Object is an outer Object, what means, that you have to declare it before you declare the class itself (class protocoll2 stands before class ServerCon). Very important is the extension Serializable, because only serializable objects can be transmitted via socket connections. The rest is very easy, in the program you must create an object of this type to interact with. (f.e. `display = new protocoll2();`).Here we need three objects (display, link and link_old). Some of them are declared in the beginning of the class others like link are declared with creating them, I wrote about this method already some paragraphs before. Now I will explain one run:

```
protocoll2 link = (protocoll2)inStream.readObject();
```

reads an object and converts it to the predefined type protocoll2, afterwards it is compared with the last reading operation from the applet, the data of the last operation are stored in link_old:

```
if
((link_old.command.equals(link.command))&&(link_old.unit.equals(
link.unit)))
```

If they differ, they are cracked to strings end are sent to the C++ server (Only set or reset commands are sent there, because this server doesn't need all the other com-

mands). The C++ Server waits for a certain length of the string, therefore we had to sent two spaces after the “set” command which is 2 characters shorter as reset. The link.unit is always of the same length. Afterwards the program waits, till this second server returns data, which should contain the performed operation code:

```
c_out.print(link.unit+link.command+"  ")
c_out.flush();
s = in.readLine();
```

The returned code, is put together to an object which is called “display” and sent back to the applet. Before starting a writing operation on an Objectstream, you have to reset the stream, that’s very important but nearly described anywhere:

```
outStream2.reset();
outStream2.writeObject(display);
```

This was already the whole Java server Application. In the next paragraph I will first explain the communication part of the applet and then explain the rest of the applet code.

6.3 Frontend Applets

The second part of the JAVA group are several applets loaded into the browser of the remote PC. Their need is to show any information available of the Fieldbus and to interact with different parts of the P-Net controlled plant.

All these applets are loaded via a HTML page, which is located on the PC having connection to the Fieldbus and a Webserver installed onto it. A Webserever is a small program running in the background. Each PC connected to the internet gets an IP Adress, is a name server installed in the system, you can give this numerical address a name. f.e. pc191.ict.tuwien.ac.at. If a Webserver is running on this PC, you get a directory on your local disc, which you can see with the browser by entering the PCs IP.

We don’t want somebody to download a program from our Webserver, we only want that they download our HTML-page. In this HTML page you can define areas, like you define a picture, in which an applet loaded from the Webserver, too, is running. The applet is loaded direct into the browser and executed, without being stored on the remote machine. The following HTML-instruction loads and starts a JAVA-applet:

```
<applet code="Display_2.class" name="Display" width=370 height=540> Sorry, no display without Java.</applet>
```

Our HTML file contains three instructions to load applets (Display_2, BottonTest and Connect), the name (here “Display”) is important for the communication between the applets. The “code” is the name of the precompiled JAVA file.

We use the communication between two applets to update variables in the Display_2 applet from other applets. So we need only one applet (Display_2) which communi-

cates with the server. Connect and Buttontest send their instructions to Display_2 where they are sent to the server. This mechanism will be described later.

First we want to have a look on some speciabilities of applets an explain the Client-Server-communication between Display_2 and ServerCon.

6.3.1 Display_2

An applet needs a special library called java.applet.*. java.awt.* is used to create visualization objects. java.awt.event.* is used for interaction with this objects, like pressing a button. All this is explained later. A applet must be declared as runable, therefore you have to put some extra parameters to the class:

```
class protocoll2 implements Serializable{

Thread t = null;

public void init() {
...
}
public void start() {
    if (t == null) {
      t = new Thread (this);
      t.start();
    }
  }
public void run()  {
...
}
}
```

An applet has a minimum of three procedures, start() creates a new runable thread, init() is only executed for one Time before run(), which exists for each thread, is executed. This is a simple skeleton which stands behind each applet used here, procedures executed by init() or run() are written before their appearance in one of this two procedures.

The Display_2 applet discussed first is the elementary applet and contains the communication to the Server PC. Therefore we will explain now all functions used for this communication:

```
public void run()  {

Socket socket=null;
ObjectOutputStream  outStream=null;
ObjectInputStream inStream2=null;
...
while(true)  {
...
if ((!connect)&&(control.command.equals("connect"))) {
```

```
        try {
          ReactRatio = control.data2;
          socket = new Socket (control.unit, control.data1);
          outStream  =   new  ObjectOutput-
Stream(socket.getOutputStream());
          inStream2 = new ObjectInput-
Stream(socket.getInputStream());
          setVisible(true);
          connect = true;
...
if (connect) {
          try {
            link.command = control.command;
            link.unit = control.unit;
            link.data1 = control.data1;
            link.data2 = control.data2;
            outStream.reset();
            outStream.writeObject(link);
            outStream.flush();
            protocoll2 display = (protcoll2)inStream2.
                                          readObject();
           ...
Thread.sleep(ReactRatio);
...
if (control.command.equals("disconnect")) connect = false;
...
if
((ctrol.command.equals(control_old.command))&&(control.unit.equa
ls(control_old.unit))) {} else
{mess4_diff=true;control_old.command = con-
trol.command;control_old.unit = control.unit;}
...
```

The above shown syntax is a part of class Display_2, all lines are standing in procedure run, that means that they are executed for each thread. The first three lines are executed ones, they declare a socket and two streams. The rest stands in a endless loop. In this loop are different ways, a Boolean called “connect” indicates if there is already a connection to the JAVA Server or not, if not and a later described object called control has a .command-string which equals “connect”, a socket with a port Number specified in .data2 and a IP Adress of a ServerSocket of value .unit, both are parts of the object control, is created. “ReactRatio” which gets it’s value here ,too, is needed to make a temporary break into this endless loop. Therefore we use the instruction Thread.sleep(xx). The creation of the streams here done, too, we already discussed with the server. “connect” is set to true and with the next loop the program will run into a different way.

If connect true, all variables of the object control are copied to the link object, this object was already mentioned with the server. Control has the same structure, but is an inner object, what means, that it cannot communicate with servers via sockets, but it’s good to communicate with other applets. After copping data, link is sent to the server, don’t forget to reset and flush the stream. Now the program stops and waits

until the display object, same type as link, is received from the server. This will happen after the command is executed at the Fieldbus.

You see that there are two more if-conditions for the control.command, disconnect is very easy, it only changes the Boolean connect, but the last one is a bit more tricky. If two equal control objects follow, a Boolean called mess4_diff is set, this indicates, that in the graphic layout, where all special painting methods has to be done, a special way of painting for two equal instructions is gone. But it's to complicate to explain at this moment, we will show it with the graphics mode. The rest here is only a coping of the new object to the old object. The next few lines will show all the rest about control and control_old:

```
public class protocoll {
    public String command="disconnect";
    public String unit="";
    public int data1=0;
    public int data2=0;
    }
public void init() {

    link = new protocoll2();
    control = new protocoll();
    control_old =new protocoll();
```

The last shown syntax explains the structure of protocol which is the basis type of control and control_old. Both created in the init() procedure, link with the protocoll2 type, that's the outer object, already explained with the server, is created here ,too. Protocoll2 is to be found over class Display_2 in the syntax and not shown here again.

What is still to say about the control object? control.command is in some cases a report line, because .command and . unit are always shoe in a status line in the graphics layout, if an error occurs or a command like "online" or "offline" expires, which has no reason for Display_2, control is not send to server and the client won't wait for the servers answer, instead it will immediately show control in the status line. How to give all control parameters the right value will be explained with ButtonTest and Connect, because these two applets are able to involve control, how will be shown with their explanation.

The second big part of this program controls the visualization. Therefore a picture is loaded into the background and with various paint instructions symbols and messages are painted onto it. This has to be done each Time something changes, we programmed it in a way, that the Backgroundpicture is only painted ones, and only the paintings onto it are done for each loop of the endless (while(true)) loop of run() procedure. If you change the browser window, f.e. you put it in the background and then into front again, you have to repaint, too.

In the begin of Display_2 are lots of defines f.e BACKGROUND_C for the Backgroundcolor, the next section contains Booleans which indicate the status of the painted symbols, f.e. Pump_status. In the run() procedure, if connect = TRUE and

after making dataexchange with the server, you will find lots of conditions, which check the value of the incoming object and set these above mentioned status indicators.

```
if ((dis-
play.command.equals("set"))&&(display.unit.equals("P1")))
Pump_status=true;
```

The procedure displayDrawings (Graphics g) contains all drawings been done with the universal graphicssymbol g. All Paintoperations depend on the former explained status Boolean, symbols like rectangles or ovals indicate the valves, the pump, the heater and the mixer, which can be switched on or off (indicated with different colors):

```
if (Heater_status)  {
      g.setColor(OPENVALVE_C);
    }
    else g.setColor(CLOSEVALVE_C);
    g.fill3DRect(98, 176, 15, 10, true);
```

This syntax shows f.e. how this works with the heater. A very special drawing indicates the filling status of tank 2. This part has an error-indication for to big values, if something occurs, the Fillstatus is painted in a different color. The status bar is a blue rectangle on a white background (color of Backgroundpicture), if the value gets bigger the bar gets longer, and the rectangle longer, too. So you would have to paint a new rectangle above the old one. But if the value is getting smaller, you would have to paint a new smaller blue bar above the big old one with the same color, nobody would see the new one, because the white background of the picture is over painted with the old, big bar. The picture is only painted ones in the beginning. Therefore we paint a white rectangle, bigger then the biggest blue bar before we paint the blue bar. This works fine if the value changes, but if the value stays equal you would see a short white blinking, so we created a Boolean (mess3_diff) to indicate if the value is equal and we only paint the white rectangle if the value is not equal:

```
try  {
H2OHeight = (int)(Weight_status*10);
Color TankColor = new Color (102,153,255);
if(mess3_diff) {g.setColor(TankColor);g.fillRect(H2Ox,H2Oy-
maxH2OHeight,H2OWidth,maxH2OHeight);mess3_diff=false;}
g.setColor(H2O_C);

if (H2OHeight>maxH2OHeight)  {
        H2OHeight = 57;
        g.setColor(Color.red);
        control.command = "error";control.unit="TANK2";
      }
}  catch (StringIndexOutOfBoundsException e) {}
   catch (NumberFormatException e)  { control.command = "error:
";control.unit="TANK2"; H2OHeight =50;}
```

```
g.fillRect(H2Ox, H2Oy-H2OHeight, H2OWidth, H2OHeight);
g.setColor (fg);
```

The same mechanism is used with the messages in the status bar and the values of temperature and weight written into the graphics. Here mess1- mess4_diff indicate the difference of values and paint if necessary a white bar, instead of a rectangle, the values are painted with the drawString command. A special condition makes it possible that only commands and status changes are indicated if the system if online (connected to the server) or if the following commands: disconnect or error occur:

```
if ((con-
nect)||(control.command.equals("disconnect"))||(control.command.
equals("error"))
```

All paint operations have x-, and y-coordinates which a relatively to a starting Point at the left top of the Graphicspanel. The values of x and y are pixels, their maximum value is given with the dimensions of the applet in the HTML description.

There is one special procedure to draw the background picture, but before it is possible to execute it, you have to load the picture into a variable of type Image which is called Background_P in our example. This loading operation can take a long Time, because it depends on the ratio of the datatransfer. If you go on executing the program before the picture is loaded completely, you will never get a full picture, because this operation is part of the init() procedure and only performed once. There is a special mechanism to wait till the picture is loaded completely:

```
try  {
    MediaTracker LoadCont = new MediaTracker(this);
    Background_P = getImage(getCodeBase(), "Pnet_5.gif");
    LoadCont.addImage(Background_P,0);
    LoadCont.waitForID(0);
  } catch (Exception e)  { control.command = "error: ";control.unit=e.toString();}
```

A Mediatracker performs such operations. You only have to create one, add the image to its LoadCont and wait for “ ready” of the loading ID. Now the picture is loaded completely and you can always get it with the following procedure:

```
private void displayImage (Graphics g)  {
   g.drawImage (Background_P, 0, 0, this);
 }
```

After we have explained how to draw pictures and symbols, we will show how to make them visible. There are to different methods. The command repaint(), we use it

at the end of the run() procedure if a new display object was sent from the server, executes the update(graphics g) method:

```
public void update(Graphics g)  {
    if (firstPaint) {  displayImage(g);firstPaint = false;}
    displayDrawings(g);
  }
```

We made a special Boolean to indicate if it's the first or an other call, because we only wanted to paint the picture once, otherwise the screen would flicker to much. If there is a change in the browser window (getting to foreground, changing position or size, ...) the paint method is automatically called up. This method always draws the picture and the graphics. There are some instructions left which are part of the init() procedure:

```
setLayout (new FlowLayout() );
Font f = new Font ("TimesRoman",Font.PLAIN,15);
setFont (f);
setVisible(true);
```

You need them to make a display possible. The Layout is important to place predefined graphics (buttons, arrays, ...) into your surface, this will be explained more in the next chapter, because it's only used there. You have to set a default font here, which is used if no specific one is defined with a draw operation or a placement of a graphics object. Very important is setVisible(true), otherwise you would see nothing.

6.3.2 Connect & ButtonTest

Both applets are able to invoke the value of the command object variables. Buttontest has a row of Buttons on it's surface to switch pump, mixer, heater and the valves. Connect has three text arrays for IP-Adress and Port of the server and the reaction ratio of the system, defined with a thread.sleep (ReactionRatio) in Display_2 run() procedure.

They both have the same mechanism to invoke the command object, the inner class protocol is component of both applets, you can find it in Display_2, too. So there are the same objects in all three applets, that's good but not necessary. Because with the following syntax it is possible to address variables in a different applet:

```
Display_2 resultApplet;

public void init() {

AppletContext browser = getAppletContext();
resultApplet = (Display_2) browser.getApplet("Display");
```

Display_2 is the name of the .class File, Display is the applets name defined in the HTML Code, both must not be the same as you see here. There is only one big rule, if you want to invoke from applet B (here ButtonTest) variables in applet A (here Display_2) you always have to load applet A, before loading applet B. It is not possible to get invoke variables of B from A. A problem is, that the browser loads the applets as placed in the HTML Code and starts at the left top corner of the window, therefore applet B is always left or below applet A. The following lines show how the invoke variables:

```
resultApplet.control.command = control.command;
resultApplet.control.unit = control.unit;
resultApplet.control.data1 = control.data1;
resultApplet.control.data2 = control.data2;
```

All variables of object command standing on the left side belong to the Display_2 command object, the other ones on the right side are members of ButtonTest's command object. The same method is used with Connect.

The ways how the command object gets it's value are different with Connect and Buttontest. First I want to explain how it works with Buttontest:

The Layout is a Gridlayout, here you can specify into how many rows and columns you would like to divide the surface, they all have to be of the same size. If you want different sizes, you have to make e level two layout in one cell and divide it again. Here it's not necessary to do so. We only have two graphical objects, buttons and canvas, which are areas with no function (therefore they have variables named space1-16). All definitions of the Layout must be done in the init() procedure as described followed:

```
private Canvas space1,space2,space3,space4,

public void init() {

Color backColor = new Color (102,102,102);
setBackground (backColor);
setLayout (new GridLayout (16,2) );
space1 = new Canvas();
space2 = new Canvas();
 ...
Button b1 = new Button("on");
    b1.addActionListener (this);
    b1.setActionCommand("b1");
    add(b1);
 Button b2 = new Button("off");
    b2.addActionListener (this);
    b2.setActionCommand("b2");
    add(b2);
 add(space1);
 add(space2);
 Button b3 = new Button("on");
```

```
...
```

As described above, you first have to create a graphical object, as each other object, then give some specifications to it and finally add it to the Layout. The Gridlayout is filled row by row. If you want to define a special Color as we do here, you can define it best with it's three primary colors as it is done in the HTML code, too. That makes it very easy to give applets and HTML pages the same color. The values of red, green and blue are between 0 and 255.

Actionlisteners are needed to indicate a performance done by a button, normally the given Command is the label you can find on the top of the bottom. Here we have more then one button with the same label, to differ them you have to give a specific Command to each, done by .setActionCommand, how to work with an event performed by pressing a button is shown next:

```
public void actionPerformed (ActionEvent e) {
    String s = e.getActionCommand();
    if (s.equals("b1")) {
      control.unit = "V1";
      control.command = "set";
    }
    ...
    resultApplet.control.command = control.command;
    resultApplet.control.unit = control.unit;
    resultApplet.control.data1 = control.data1;
    resultApplet.control.data2 = control.data2;
```

A special procedure called "actionPerformed" that will only be performed if one ActionListener indicates a performance, contains if-conditions to look which special action was performed. Each action has it's own command defined with building the buttons. Afterwards the specific value of control.command and control.unit is given to the same object in the resultApplet (Display_2), this resultApplet was already defined in the init() procedure.

As you can see there is no run() procedure in the class, that means that nothing is executed if no button is pressed.

The Connect applet is not very different, there is the same Gridlayout, the same inner Class Protocol and the same result Applet. But as you see, the Gridlayout has only one column, so we have define a sublayout for the to buttons at the top:

```
northPanel = new Panel();
northPanel.setLayout (new GridLayout(1,2));
...
add(northPanel);
```

We define a panel, something you can put graphical Objects into, call it northPanel and define a new Layout for it. Here one row with two columns. Afterwards we put all Objects into it, with the add function as described with Buttontest. After finishing we have to put the panel to the first Layout. Attention here is only one space needed for

an empty row because there is only one column. A new not used before element is a TextField, if you want to get the String of this field use TextField.getText(). .parseInt(String) converts the String to an integer value:

```
ServerName = new TextField("", 20);
```

These were the main structures of Connect and ButtonTest, how to work with the applets and which Strings must be entered is explained in the chapter called “Manual”.

7 Conecret Solutions

7.1 Comunication protocol VC++ Server - JAVA Client/Server.

We tried to create the VC++Server - JAVA Client/Server connection as easy as possible. The JAVA applet - JAVA Client/Server uses objects for the communication, the VC++-JAVA connection just use strings.The protocoll is very simple, a few commands are implemented:

```
V1set,V1reset, V2set, V2reset, V3set, V3reset, V4set, V4reset,
V5set, V5reset, P1set, P1reset, M1set, M1reset
```

A request from the JAVA applet will be processed from the VC++ server in a very simple way:

The commant „set vent one“ comes to the JAVA Server/Client from the applet, the JAVA programme transforms this command to the string „V1set“ and send it to the VC++ Server. This programme was waiting for a command (in the online mode). The VC++ programme analyzes the command and executes it. (create a VIGO object, link it to the P-NET unit (f.e. Vent 1), change the right property (f.e. set flag regsiter 7) and dislink the object). After that the VC++ programme sends the command back to the JAVA server to confirme the execution. The VC++ programme makes an update of the display, but user input will not be recognized.

When the VC++ server doesn't send back the command the JAVA program waits forever. A time unit should be implemented to avoid such a situation but it never happend at the laboratory.

The real time reaction of the system was fine, the time distance between pressing a button at the applet an getting the conformation was about half a secound.

7.2 The OLE-VisualC++ Server

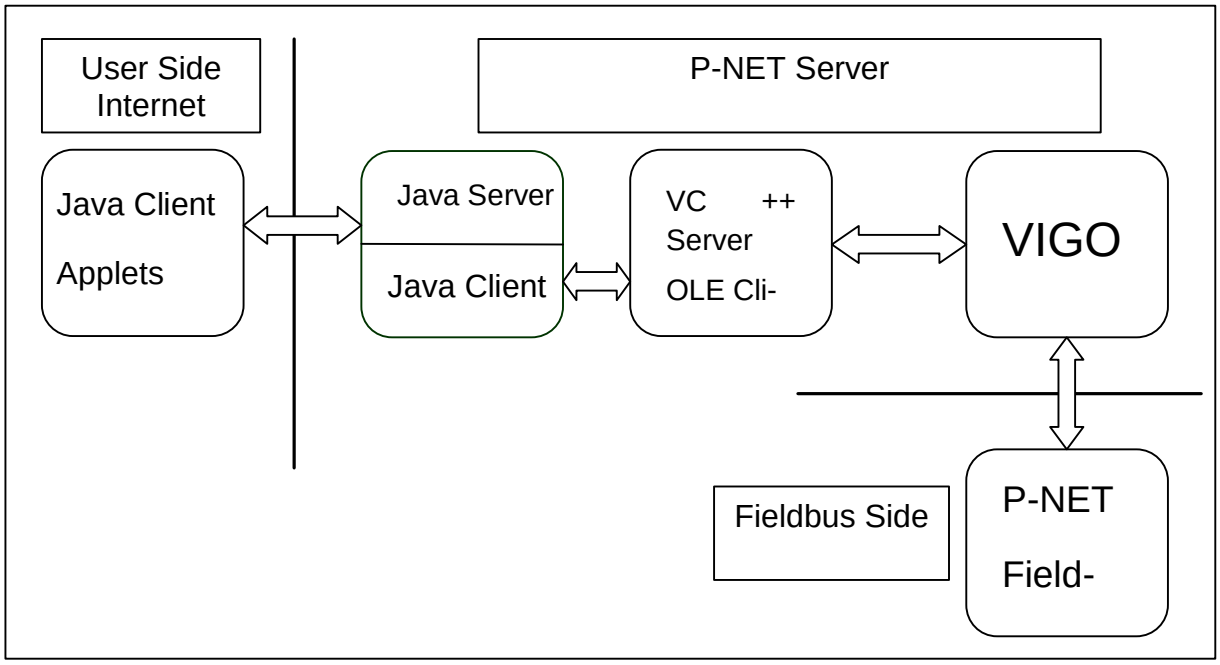
We tried to include this part of the project into the JAVA server, but that was not so easy because of some problems, described a few pages before. The main advantage of VisuaC++ in our case is that it is easy to handle with the OLE interface. To create a VIGO object in VisualC++ only three commands are necessary:

```
vobj= new IvigoPro;
vobj->CreateDispatch("Vigo.Pro");
vobj->SetPhysId("Waage.Digital_IO_1.FlagReg[7]");
```

one JAVA Client.

And with the help of the MFC development enviroment it was easy to create a little OLE-VC++ Server.

But there was another consequence, now we have two server programs running at the same host and Picture 7.2.1 shows the structrue of the three programs:



Picture 7.1 Current Comunication Solution

The VC++ Server programme can run in two ways:

- **offline mode**: the P-NET can be controlled with the VC++ program.
- **online mode**: the P-NET is controlled from the JAVA applets, the VC++ program ist just a the bridge between JAVA and VIGO, the user can't controll the P-NET with the VC++ server.7

After the startup the programme is in the offline mode (displayed at the J-ServerMessage line), the P-NET can be controlled from the user, all sensores are displayed. Before the program can run in online mode the JAVA server has to be started in a seperate dos window (c:\ java ServerMain). After that you can press the „Start C-OLE-Java Server“ Button and after a little while the connection will be confirmed at the J-Server Message line. Now the user can controll the P-NET over the JAVA front end.

One problem of writing this programme was this: When you try to create a seperate VIGO object for each of the 27 P-NET objects of the ICT P-NET you will need very much system memory. I tried it at the ICT PC (Pentium, Windows NT, 64 MB RAM), and the program could not be started, 64 MB RAM semmed to be to little.So i created just one VIGO Object, that means that before you can use it for a certan P-NET object you have to link it to this object, and after using the object you have to dislink the object. f.E.:

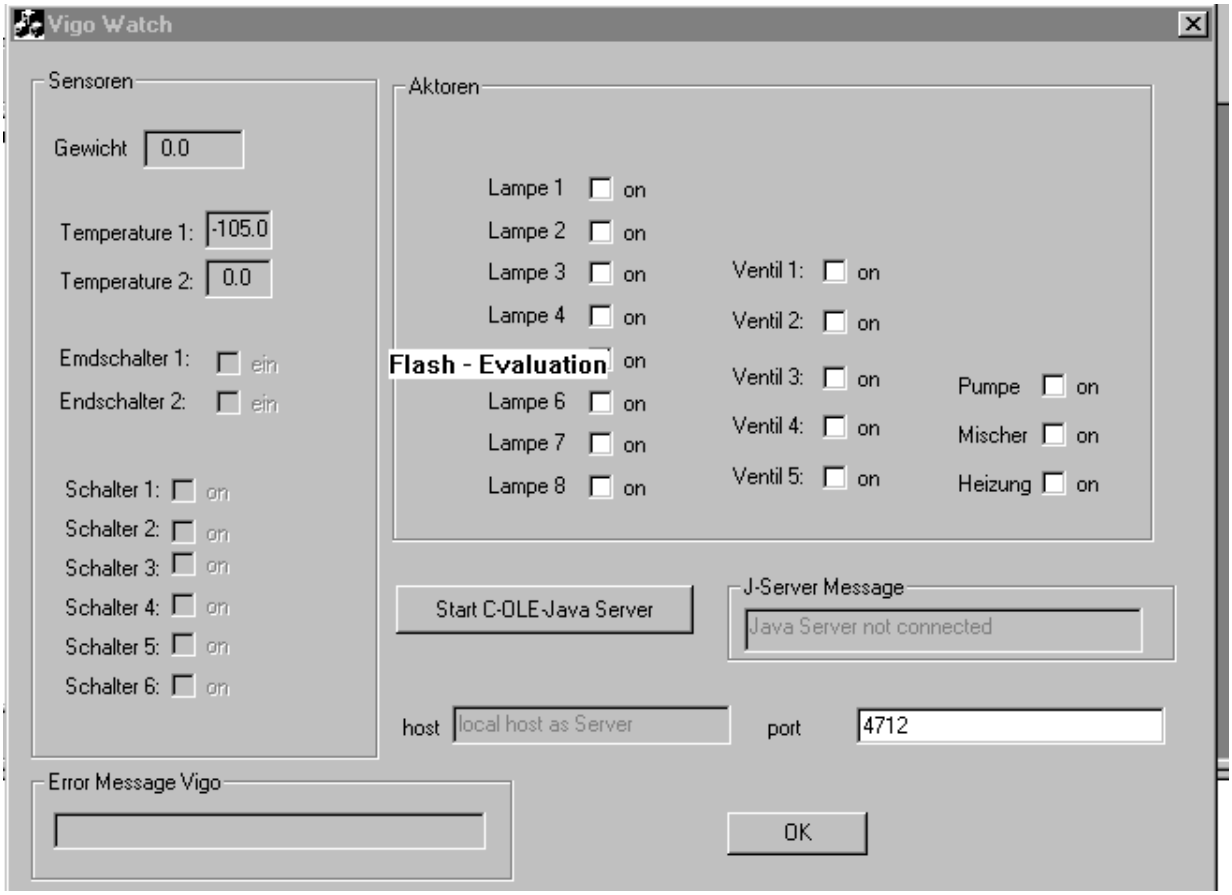
```
vobj->CreateDispatch("Vigo.Pro");
```

```
vobj->SetPhysId("Waage.Digital_IO_1.FlagReg[7]");
use the object
vobj->ReleaseDispatch();
```

That works fine, but the problem is that it is a little bit slow. So a posssible solution for this problem would be to create several VIGO objects for the most used P-NET objects. This could make the programme little bit faster.

To close the VC++ server program press the OK button.

The VC++ programme is written in Visuac C++ form Mircosoft, Vers. 5.0. Be carefule, when you try to compile it with Visual C++ Ver. 6.0 you will get many compiler erros. The reason ist that the type VARIANT which is necessary for the OLE interface is implementet in a different way (in vers. 6 the VARINAT tag BOOL does not exist). To communicate with VIGO you need the version 5.0.



Picture 7.2 Frontende of C++ Program

8 Manual

To make the system work you have always follow a certain way. The C++ Server is an .exe File you can find in the Homedirectory, the JavaServer are two .class Files called ServerMain.class and ServerCon.class, you have to start ServerMain, because ServerCon is a Thread which is started with each communication request from the foreign Browser. There are three more .class Files called Display_2.class, Connect.class and ButtonTest.class, these files are applets needed with the HTML-Page lab.html. This page also needs the backgroundpicture Pnet_5.gif and all images in the subdirectory /pictures.

There are to speciabilities, to make all the applets HTML-pages and pictures loadable, the Homedirectory must be a Webserver directory, we used the Xtami freeware Webserver, see the appendix for more information. This Webserver has a subdirectory called WebPages, all directories which you can find in this directory are accessible with the Internet.

To run ServerMain and ServerCon you need a VirtualMachine, because JAVA Files are only precompiled to make it runable on each platform. The Windows Virtual Machine needs some specific settings, which you can find in their Manual pages or you can have a look at our autoexec.bat in the appendix. To run the File enter „java ServerMain“ into the command prompt. Attention, with WindowsNT, you have always to execute autoexec.bat first after opening a new command prompt.

8.1 The C++ Server

The only importend thing is, that you have to start the Server before connecting the applet, but you have to connect to the Java Server afterwards. Please do not hurry, cause it takes some time after starting the Server and after connecting, too.

8.2 The Java Server

There is no restriction which Server must be started first, but both Servers must run before a foreign communication partner connects to the system. The JAVA Server has no Frontend and therefore no possibilities to interact wrong or right.

8.3 The Browser Applets

To load the applets, you have only to enter the URL of the Server on which the former described WebServer is running and load lab.html. There are to interactive pads on the left side of the page, the Buttons on the upper pad are used to switch valves, heater, pump and mixer and only work if you are connected.

To connect you have to enter the JAVA Servers ServerName, the right port (4711) and a value of the reactionRatio. The reactionRatio performs a break of the given Number of milliseconds. On normal fast PCs you can set this value to zero. With pressing the connect Button the Server gets connected. A status message is show

below the plant picture, this status bar shows each interaction, but not until the requested changed status is shown, in the plant picture the movement is shown after the Fieldbus has executed it.

 © ICT, 1999

Appendix

Listings JAVA

ServerMain.java

```
//  ****************************************************************
//  *                                                              *
//  *   Appletname:    ServerMain.java                             *
//  *   Autor:         Roland Steiner, Stefan Hutter               *
//  *   Date:          03.99                                       *
//  *   Description1:  Application to comunicate with applets      *
//  *                  and a second C++ Server. Both comunications *
//  *                  are socket comunications, one with strings the*
//  *                  other with objects, ServerCon translates these*
//  *                  differnet formats to interact. This program only*
//  *                  starts a Thread for each connection to do the *
//  *                  above mentioned things. the Thread is called  *
//  *                  ServerCon.                                  *
//  *                                                              *
//  ****************************************************************

import java.net.* ;

public class ServerMain {
  public static void main (String[] args) {

    ServerSocket server;
    Socket s = null;
    ServerCon doIt;
    int port = 4711;

    try {
      server = new ServerSocket (port);
      System.out.println("Server running....");
      while(true) {
        s = server.accept();
        System.out.println("Client online");
        doIt = new ServerCon (s);
      }
    } catch (Exception e) {
        System.out.println("ServerMain " +e);
        System.exit(1);
    }
  }
}
```

ServerCon.java

```
//  ****************************************************************
//  *                                                              *
//  *   Appletname:    ServerCon.java                              *
//  *   Autor:         Roland Steiner, Stefan Hutter               *
//  *   Date:          03.99                                       *
//  *   Description1:  Application Thread to comunicate with applets *
//  *                  and a second C++ Server. Both comunications *
//  *                  are socket comunications, one with strings the*
//  *                  other with objects, ServerCon translates these*
//  *                  differnet formats to interact.              *
//  *                                                              *
```

```
//  ****************************************************************

import java.net.* ;
import java.io.* ;
import java.util.* ;

class protocoll2 implements Serializable{
  public String command="";
  public String unit="";
  public int data1=0;
  public int data2=0;
}

public class ServerCon implements Runnable {

  private Socket socket,socket_c;
  private PrintWriter out;
  BufferedReader in;
  private Thread t;
  public protocoll2 display,link_old;
  PrintWriter c_out;
  String s;

  public ServerCon (Socket s) {
    socket = s;
    t = new Thread(this);
    t.start();
  }


  public void run() {

      display = new protocoll2();
      link_old = new protocoll2();

      try {

        System.out.println("Trying to connect c++ Server......");

        socket_c = new Socket ("",4712);

        System.out.println("done");

        System.out.println("Reading data.............");
        ObjectInputStream inStream = new ObjectInput-
Stream(socket.getInputStream());
        ObjectOutputStream outStream2  =   new  ObjectOutput-
Stream(socket.getOutputStream());

        c_out = new PrintWriter(new BufferedWriter(new OutputStream-
Writer(socket_c.getOutputStream() ) ) );
        in = new BufferedReader(new InputStreamReader
(socket_c.getInputStream() ) );

        while(true) {

          try{

          protocoll2 link = (protocoll2)inStream.readObject();

          display.data1 = 0;
          display.data2 = 0;
          display.command = "not ready";
          display.unit = " not defined";
```

```
          if
  ((link_old.command.equals(link.command))&&(link_old.unit.equals(link.unit)))

          { display.data1=1; }

          else

          {   System.out.print("New command ");
              if((link.command.equals("set"))||(link.command.equals("reset")))
                {
                  System.out.print("Write ( "+link.command+link.unit+" ) to c++
  server...");
                  if(link.command.equals("set")){System.out.print("writing
  ");c_out.print(link.unit+link.command+"  ");}
                  if(link.command.equals("reset")){System.out.print("writing
  ");c_out.print(link.unit+link.command);}
               System.out.print("done now waiting for data....");
             c_out.flush();
              s = in.readLine();
                  System.out.println("done. String: "+s);
                  if(s.equals("V1set  ")){dis-
  play.command="set";display.unit="V1";}

  if(s.equals("V1reset")){display.command="reset";display.unit="V1";}
             if(s.equals("V2set  ")){display.command="set";display.unit="V2";}

  if(s.equals("V2reset")){display.command="reset";display.unit="V2";}
             if(s.equals("V3set  ")){display.command="set";display.unit="V3";}

  if(s.equals("V3reset")){display.command="reset";display.unit="V3";}
             if(s.equals("V4set  ")){display.command="set";display.unit="V4";}

  if(s.equals("V4reset")){display.command="reset";display.unit="V4";}
             if(s.equals("V5set  ")){display.command="set";display.unit="V5";}

  if(s.equals("V5reset")){display.command="reset";display.unit="V5";}
             if(s.equals("M1set  ")){display.command="set";display.unit="M1";}

  if(s.equals("M1reset")){display.command="reset";display.unit="M1";}
             if(s.equals("P1set  ")){display.command="set";display.unit="P1";}

  if(s.equals("P1reset")){display.command="reset";display.unit="P1";}
             if(s.equals("H1set  ")){display.command="set";display.unit="H1";}

  if(s.equals("H1reset")){display.command="reset";display.unit="H1";}
                  }
          }

          c_out.flush();
          outStream2.reset();
          outStream2.writeObject(display);
          link_old = link;

           }  catch(ClassNotFoundException e){ System.out.println(e); }
         }
       } catch (Exception e) { System.out.println("ServerCon " + e);}
    }
  }
```

Display_2.java

```
  //  ************************************************************
  //  *                                                          *
  //  *   Appletname:    Display_2.java                          *
  //  *   Autor:         Roland Steiner, Stefan Hutter           *
```

```
//  *   Date:          03.99                                          *
//  *   Description1:  Main Applet for displaying the information,    *
//  *                  and seting up the tcp/ip connection. Displays *
//  *                  data from server and delivers data from other *
//  *                  applets to the server.                         *
//  *                                                                 *
//  *******************************************************************

import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;

//  *******************************************************************
//  *                                                                 *
//  *   class protocoll2                                              *
//  *                                                                 *
//  *   Description1:  Serializable class for object sendind via tcp/ip*
//  *                  is set with equal values as prtocoll1, an inner *
//  *                  class of class Display_2 to interact with the   *
//  *                  other applets of this programm.                *
//  *                                                                 *
//  *******************************************************************

class protocoll2 implements Serializable{
  public String command="";
  public String unit="";
  public int data1=0;
  public int data2=0;
}

//  *******************************************************************
//  *                                                                 *
//  *   class Display_2                                               *
//  *                                                                 *
//  *******************************************************************

public class Display_2 extends Applet  implements Runnable{

  public protocoll2 link;
  public protocoll control;
  public protocoll control_old;

  Thread t = null;

  public class protocoll {
    public String command="disconnect";
    public String unit="";
    public int data1=0;
    public int data2=0;
    }

  public static final Color  BACKGROUND_C = Color.gray;
  public static final Color  H2O_C = Color.blue;
  public static final Color  OPENVALVE_C = Color.green;
  public static final Color  CLOSEVALVE_C = Color.red;
  public static final Color  HEATER_C = Color.red;

  public boolean Pump_status = false;
  public boolean Heater_status = false;
  public boolean Mix_status = false;
  public boolean[] V_status =   { true, true, true, true, true ,true};
  public double Temp1_status = 20;
  public double Temp2_status = 20;
```

```
  public double Weight_status = 2;

  Image Background_P;

  public boolean firstPaint = true;
  public boolean connect = false;
  public boolean mess1_diff = false;
  public boolean mess2_diff = false;
  public boolean mess3_diff = false;
  public boolean mess4_diff = false;

//  ****************************************************************
//  *                                                              *
//  *   public void init()                                         *
//  *                                                              *
//  *   Description:   Creates records of the two classes protocoll and*
//  *                  protocoll2, loads the Backgroung Image with     *
//  *                  controll of a mediatracker which stops processin*
//  *                  further as long as the picture isn't loaded.    *
//  *                  Sets Layout and Font.                       *
//  *                                                              *
//  ****************************************************************

  public void init() {

    link = new protocoll2();
    control = new protocoll();
    control_old =new protocoll();

    try  {
      MediaTracker LoadCont = new MediaTracker(this);
      Background_P = getImage(getCodeBase(), "Pnet_5.gif");
      LoadCont.addImage(Background_P,0);
      LoadCont.waitForID(0);
    } catch (Exception e)  { control.command = "error:
";control.unit=e.toString();}

    setLayout (new FlowLayout() );
    Font f = new Font ("TimesRoman",Font.PLAIN,15);
    setFont (f);
    setVisible(true);

  }

//  ****************************************************************
//  *                                                              *
//  *   public void displayImage()                                 *
//  *                                                              *
//  *   Description:   Draws loaded picture.                       *
//  *                                                              *
//  ****************************************************************

  private void displayImage (Graphics g)  {

    g.drawImage (Background_P, 0, 0, this);
  }

//  ****************************************************************
//  *                                                              *
//  *   public void displayDrawings()                              *
//  *                                                              *
//  *   Description1:  To draw and redraw all grafics              *
//  *                                                              *
//  ****************************************************************
```

```
  private void displayDrawings (Graphics g)  {

    Color bg = getBackground();
    Color fg = getForeground();
    int ValveHeight = 10;
    int ValveWidth = 30;
    int H2Ox = 163;
    int H2Oy = 396;
    int H2OWidth = 6;
    int H2OHeight = 20;
    int maxH2OHeight = 57;

//  ****************************************************************
//  *                                                              *
//  *   Description2:  Loop to draw rectangels to indicate valve status*
//  *                  which is read from server and done each reding  *
//  *                  cycle.                                      *
//  *                                                              *
//  ****************************************************************

    for (int i = 1; i<6; i++)  {
      g.setColor(CLOSEVALVE_C);
      if (V_status[i])  {
    g.setColor (OPENVALVE_C);
      }
      switch (i)  {
        case 1 :
    g.fill3DRect(141, 198, 9, 15, true);
    break;
       case 2 :
    g.fill3DRect(141, 216, 9, 15, true);
    break;
    case 3 :
    g.fill3DRect(214, 198, 9, 15, true);
    break;
    case 4 :
    g.fill3DRect(192, 246, 15, 9, true);
    break;
    case 5 :
    g.fill3DRect(142, 438, 9, 15, true);
    break;
      }
    }

//  ****************************************************************
//  *                                                              *
//  *   Description3:  Drawing symbols for Pump, Heater and Mixer with *
//  *                  their actual status, which is done after each   *
//  *                  reading cycle from the server.              *
//  *                                                              *
//  ****************************************************************

    g.setColor(Color.black);
    g.fillOval(208, 265, 24, 24);
    if (Pump_status)  {
      g.setColor(OPENVALVE_C);
    }
    else g.setColor(CLOSEVALVE_C);
    g.fill3DRect(250, 272,15, 10, true);

    if (Heater_status)  {
      g.setColor(OPENVALVE_C);
    }
    else g.setColor(CLOSEVALVE_C);
    g.fill3DRect(98, 176, 15, 10, true);
```

```
    if (Mix_status)  {
      g.setColor(OPENVALVE_C);
    }
    else g.setColor(CLOSEVALVE_C);
    g.fill3DRect(98, 403, 15, 10, true);



//  ****************************************************************
//  *                                                              *
//  *   Description4:  Drawing part to show messages on display, but   *
//  *                  only if system is connected to server or the    *
//  *                  messages are "disconnect" or "error". If the    *
//  *                  message information differes from the old info  *
//  *                  it is necessary to clean the display with drawin*
//  *                  a recantgle in background color              *
//  *                                                              *
//  ****************************************************************


    try  {
      if ((con-
nect)||(control.command.equals("disconnect"))||(control.command.equals("error"
))) {
        g.setColor(Color.white);
        if(mess1_diff) {g.fillRect(20,65,110,22);mess1_diff=false;}
        if(mess2_diff) {g.fillRect(20,290,110,22);mess2_diff=false;}
        if(mess3_diff) {g.fillRect(20,313,110,20);}
        if(mess4_diff) {g.fillRect(54,518,316,19);mess4_diff=false;}
        g.setColor(fg);
        g.drawString(" "+Double.toString(Temp1_status)+" °C ",25,82);
        g.drawString(" "+Double.toString(Temp2_status)+" °C ",25,307);
        g.drawString(" "+Double.toString(Weight_status)+" kg ",25,328);
        g.drawString(control.command+" "+control.unit,54,534);
      }

//  ****************************************************************
//  *                                                              *
//  *   Description5:  Part of Drawing void to indicate the filling    *
//  *                  status of the tank with a bar and indicates an  *
//  *                  error if H20Height is bigger as maxH20Height    *
//  *                                                              *
//  ****************************************************************

      H2OHeight = (int)(Weight_status*10);
      Color TankColor = new Color (102,153,255);
      if(mess3_diff) {g.setColor(TankColor);g.fillRect(H2Ox,H2Oy-
maxH2OHeight,H2OWidth,maxH2OHeight);mess3_diff=false;}
      g.setColor(H2O_C);

      if (H2OHeight>maxH2OHeight)  {
        H2OHeight = 57;
        g.setColor(Color.red);
        control.command = "error";control.unit="TANK2";
      }
    }  catch (StringIndexOutOfBoundsException e) {}
       catch (NumberFormatException e)  { control.command = "error:
";control.unit="TANK2"; H2OHeight =50;}

    g.fillRect(H2Ox, H2Oy-H2OHeight, H2OWidth, H2OHeight);
    g.setColor (fg);
  }

//  ****************************************************************
```

```
//  *                                                                  *
//  *   public void paint()                                            *
//  *                                                                  *
//  *   Description1:                                                  *
//  *                                                                  *
//  ********************************************************************

  public void paint(Graphics g)  {
   displayImage(g);
   displayDrawings (g);
  }

//  ********************************************************************
//  *                                                                  *
//  *   public void update()                                           *
//  *                                                                  *
//  *   Description1:                                                  *
//  *                                                                  *
//  ********************************************************************

  public void update(Graphics g)  {
    if (firstPaint) {  displayImage(g);firstPaint = false;}
    displayDrawings(g);
  }

//  ********************************************************************
//  *                                                                  *
//  *   public void start()                                            *
//  *                                                                  *
//  *   Description1:  creates new Thread of this applet, because this *
//  *                  programm can be started for serveral times      *
//  *                                                                  *
//  ********************************************************************

  public void start() {

    if (t == null) {
      t = new Thread (this);
      t.start();
    }
  }

//  ********************************************************************
//  *                                                                  *
//  *   public void run()                                              *
//  *                                                                  *
//  *   Description1:  Action performed for each running thread        *
//  *                                                                  *
//  ********************************************************************

  public void run()  {

    Socket socket=null;
    ObjectOutputStream  outStream=null;
    ObjectInputStream inStream2=null;
    int ReactRatio = 100;

    while(true) {

//  ********************************************************************
//  *                                                                  *
//  *   Description2:  creates new socket connection with an object in-*
//  *                  and one outputstream and sets connect to true,  *
//  *                  only posible if connect is false and command    *
//  *                  equals "connect"                                *
```

```
//  *                                                                *
//  ******************************************************************

    if ((!connect)&&(control.command.equals("connect"))) {
        try {
          ReactRatio = control.data2;
          socket = new Socket (control.unit, control.data1);
          outStream  =   new  ObjectOutputStream(socket.getOutputStream());
          inStream2 = new ObjectInputStream(socket.getInputStream());
          setVisible(true);
          connect = true;
        } catch (Exception e) { control.command="error:
";control.unit=e.toString();connect = false; }
      }

//  ******************************************************************
//  *                                                                *
//  *   Description3:  Sets link (protocoll2 for server comunication)  *
//  *                  to value of control (protocoll for applet-    *
//  *                  communication) and writes this to server. It is *
//  *                  importatnt to .reset the stream before writing  *
//  *                  an object. Display is the read protocoll2 from  *
//  *                  server, which sets in several if conditions the *
//  *                  status of the drawed sybols.                  *
//  *                                                                *
//  ******************************************************************

      if (connect) {
          try {
            link.command = control.command;
            link.unit = control.unit;
            link.data1 = control.data1;
            link.data2 = control.data2;
            outStream.reset();
            outStream.writeObject(link);
            outStream.flush();
            protocoll2 display = (protocoll2)inStream2.readObject();

            if (display.command.equals("init")) { control.command =
"init";control.unit ="DISPLAY";}
            if (display.command.equals("online")) { control.command = "on-
line"; control.unit="DISPLAY";}
            if ((display.command.equals("set"))&&(display.unit.equals("V1")))
V_status[1]=true;
            if ((dis-
play.command.equals("reset"))&&(display.unit.equals("V1")))
V_status[1]=false;
            if ((display.command.equals("set"))&&(display.unit.equals("V2")))
V_status[2]=true;
            if ((dis-
play.command.equals("reset"))&&(display.unit.equals("V2")))
V_status[2]=false;
            if ((display.command.equals("set"))&&(display.unit.equals("V3")))
V_status[3]=true;
            if ((dis-
play.command.equals("reset"))&&(display.unit.equals("V3")))
V_status[3]=false;
            if ((display.command.equals("set"))&&(display.unit.equals("V4")))
V_status[4]=true;
            if ((dis-
play.command.equals("reset"))&&(display.unit.equals("V4")))
V_status[4]=false;
            if ((display.command.equals("set"))&&(display.unit.equals("V5")))
V_status[5]=true;
```

```
            if ((dis-
play.command.equals("reset"))&&(display.unit.equals("V5")))
V_status[5]=false;
            if ((display.command.equals("set"))&&(display.unit.equals("V6")))
V_status[6]=true;
            if ((dis-
play.command.equals("reset"))&&(display.unit.equals("V6")))
V_status[6]=false;
            if ((display.command.equals("set"))&&(display.unit.equals("P1")))
Pump_status=true;
            if ((dis-
play.command.equals("reset"))&&(display.unit.equals("P1")))
Pump_status=false;
            if ((display.command.equals("set"))&&(display.unit.equals("H1")))
Heater_status=true;
            if ((dis-
play.command.equals("reset"))&&(display.unit.equals("H1")))
Heater_status=false;
            if ((display.command.equals("set"))&&(display.unit.equals("M1")))
Mix_status=true;
            if ((dis-
play.command.equals("reset"))&&(display.unit.equals("M1")))  Mix_status=false;

            if (display.unit.equals("T1"))  {Temp1_status = Dou-
ble.valueOf(display.command).doubleValue();mess1_diff=true;}
            if (display.unit.equals("T2"))  {Temp1_status = Dou-
ble.valueOf(display.command).doubleValue();mess2_diff=true;}
            if (display.unit.equals("W1"))  {Temp1_status = Dou-
ble.valueOf(display.command).doubleValue();mess3_diff=true;}

//  ****************************************************************
//  *                                                              *
//  *   Description4:  Thread.sleep controls refresh ratio, because it *
//  *                  blocades the thread for the giveb ratio to the  *
//  *                  next read and write proces. Attention, the serve*
//  *                  thread also stops for this time because of waiti*
//  *                   for new data to read.                      *
//  *                                                              *
//  ****************************************************************

            Thread.sleep(ReactRatio);

          } catch (Exception e)  { control.command = "error:
";control.unit=e.toString(); connect = false;}
      }
    if (control.command.equals("disconnect")) connect = false;
    if ((con-
trol.command.equals(control_old.command))&&(control.unit.equals(control_old.un
it))) {} else {mess4_diff=true;control_old.command = con-
trol.command;control_old.unit = control.unit;}

    repaint();
    }
  }
}
```

ButtonTest.java

```
//  ****************************************************************
//  *                                                              *
//  *   Appletname:    ButtonTest.java                             *
//  *   Autor:         Roland Steiner, Stefan Hutter               *
//  *   Date:          03.99                                       *
```

```
//  *   Description:   Applet to invoke status of certain states of *
//  *                  the syteme, which interactes with the main   *
//  *                  applet ( Display_2.java ) via a common class *
//  *                  called protocoll with the resultApplet function *
//  *                                                               *
//  *****************************************************************

import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class ButtonTest extends Applet
  implements ActionListener {

  private Canvas
space1,space2,space3,space4,space5,space6,space7,space8,space9,space10,space11
,space12,space13,space14,space15,space16;
  private Label mess;

  Display_2 resultApplet;

  public protocoll control;

  public class protocoll {
    public String command;
    public String unit;
    public int data1;
    public int data2;
    }

  public int farbe;

//  *****************************************************************
//  *                                                               *
//  *   public void init()                                          *
//  *                                                               *
//  *   Description1 : creates resultApplet, the Layoutcomponents and  *
//  *            the new class control                              *
//  *                                                               *
//  *****************************************************************

  public void init() {

    control = new protocoll();
    control.command = "start";
    control.unit = "";
    control.data1 = 0;
    control.data2 = 0;

    AppletContext browser = getAppletContext();
    resultApplet = (Display_2) browser.getApplet("Display");

    Color backColor = new Color (102,102,102);
    setBackground (backColor);

    setLayout (new GridLayout (16,2) );

    space1 = new Canvas();
    space2 = new Canvas();
    space3 = new Canvas();
    space4 = new Canvas();
    space5 = new Canvas();
    space6 = new Canvas();
    space7 = new Canvas();
    space8 = new Canvas();
```

```
        space9 = new Canvas();
        space10 = new Canvas();
        space11 = new Canvas();
        space12 = new Canvas();
        space13 = new Canvas();
        space14 = new Canvas();
        space15 = new Canvas();
        space16 = new Canvas();

    //  ****************************************************************
    //  *                                                              *
    //  *   Description2:  It is important not only to add an ActionListene*
    //  *                  but also to set the ActionCommand to a different*
    //  *                  value, default is the value of the buttonn   *
    //  *                  surace, but here they all would have the same.  *
    //  *                                                              *
    //  ****************************************************************


        Button b1 = new Button("on");
        b1.addActionListener (this);
        b1.setActionCommand("b1");
        add(b1);
        Button b2 = new Button("off");
        b2.addActionListener (this);
        b2.setActionCommand("b2");
        add(b2);
        add(space1);
        add(space2);
        Button b3 = new Button("on");
        b3.addActionListener (this);
        b3.setActionCommand("b3");
        add(b3);
        Button b4 = new Button("off");
        b4.addActionListener (this);
        b4.setActionCommand("b4");
        add(b4);
        add(space3);add(space4);
        Button b5 = new Button("on");
        b5.addActionListener (this);
        b5.setActionCommand("b5");
        add(b5);
        Button b6 = new Button("off");
        b6.addActionListener (this);
        b6.setActionCommand("b6");
        add(b6);
        add(space5);add(space6);
        Button b7 = new Button("on");
        b7.addActionListener (this);
        b7.setActionCommand("b7");
        add(b7);
        Button b8 = new Button("off");
        b8.addActionListener (this);
        b8.setActionCommand("b8");
        add(b8);
        add(space7);add(space8);
        Button b9 = new Button("on");
        b9.addActionListener (this);
        b9.setActionCommand("b9");
        add(b9);
        Button b10 = new Button("off");
        b10.addActionListener (this);
        b10.setActionCommand("b10");
        add(b10);
        add(space9);add(space10);
```

```
      Button b11 = new Button("on");
      b11.addActionListener (this);
      b11.setActionCommand("b11");
      add(b11);
      Button b12 = new Button("off");
      b12.addActionListener (this);
      b12.setActionCommand("b12");
      add(b12);
      add(space11);add(space12);
      Button b13 = new Button("on");
      b13.addActionListener (this);
      b13.setActionCommand("b13");
      add(b13);
      Button b14 = new Button("off");
      b14.addActionListener (this);
      b14.setActionCommand("b14");
      add(b14);
      add(space13);add(space14);
      Button b15 = new Button("on");
      b15.addActionListener (this);
      b15.setActionCommand("b15");
      add(b15);
      Button b16 = new Button("off");
      b16.addActionListener (this);
      b16.setActionCommand("b16");
      add(b16);

      setVisible(true);
    }

//  ****************************************************************
//  *                                                              *
//  *   public void actionPerformed( ActionEvent e)                *
//  *                                                              *
//  *   Description1:  sets values of control.unit and control.command *
//  *                  for certain actions                         *
//  *                                                              *
//  ****************************************************************

    public void actionPerformed (ActionEvent e) {
      String s = e.getActionCommand();
      if (s.equals("b1")) {
        control.unit = "V1";
        control.command = "set";
      }
      else
      if (s.equals("b2")) {
        control.unit = "V1";
        control.command = "reset";
      }
      if (s.equals("b3")) {
        control.unit = "V2";
        control.command = "set";
      }
      else
      if (s.equals("b4")) {
        control.unit = "V2";
        control.command = "reset";
      }
      else
      if (s.equals("b5")) {
        control.unit = "V3";
        control.command = "set";
      }
      if (s.equals("b6")) {
```

```
        control.unit = "V3";
        control.command = "reset";
      }
      else
      if (s.equals("b7")) {
        control.unit = "V4";
        control.command = "set";
      }
      else
      if (s.equals("b8")) {
        control.unit = "V4";
        control.command = "reset";
      }
      if (s.equals("b9")) {
        control.unit = "V5";
        control.command = "set";
      }
      else
      if (s.equals("b10")) {
        control.unit = "V5";
        control.command = "reset";
      }
      else
      if (s.equals("b11")) {
        control.unit = "P1";
        control.command = "set";
      }
      if (s.equals("b12")) {
        control.unit = "P1";
        control.command = "reset";
      }
      else
      if (s.equals("b13")) {
        control.unit = "H1";
        control.command = "set";
      }
      else
      if (s.equals("b14")) {
        control.unit = "H1";
        control.command = "reset";
      }
      if (s.equals("b15")) {
        control.unit = "M1";
        control.command = "set";
      }
      else
      if (s.equals("b16")) {
        control.unit = "M1";
        control.command = "reset";
      }

  //  ****************************************************************
  //  *                                                              *
  //  *   Description2:  gives control of class protocoll of Applet      *
  //  *                  Display_2 the same value as the contol variables*
  //  *                  are in ButtonTest                           *
  //  *                                                              *
  //  ****************************************************************

      resultApplet.control.command = control.command;
      resultApplet.control.unit = control.unit;
      resultApplet.control.data1 = control.data1;
      resultApplet.control.data2 = control.data2;
    }
  }
```

Connect.java

```
//  ****************************************************************
//  *                                                              *
//  *   Appletname:    Connect.java                                *
//  *   Autor:         Roland Steiner, Stefan Hutter               *
//  *   Date:          03.99                                       *
//  *   Description:   Applet to give server information and connect   *
//  *                  the syteme, which interactes with the main  *
//  *                  applet ( Display_2.java ) via a common class *
//  *                  called protocoll with the resultApplet function *
//  *                                                              *
//  ****************************************************************

import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;


public class Connect extends Applet
  implements ActionListener {

  private Canvas
space1,space2,space3,space4,space5,space6,space7,space8,space9,space10;

  Panel northPanel;

  private TextField ServerName;
  private TextField Port;
  private TextField RefreshRatio;

  Display_2 resultApplet;

  public protocoll control;

  public class protocoll {
    public String command;
    public String unit;
    public int data1;
    public int data2;
    }

  public int farbe;

  public void init() {

    control = new protocoll();
    control.command = "start";
    control.unit = "";
    control.data1 = 0;
    control.data2 = 0;

    AppletContext browser = getAppletContext();
    resultApplet = (Display_2) browser.getApplet("Display");

    Color backColor = new Color (102,102,102);
    setBackground (backColor);

    setLayout ( new GridLayout(10,1) );

    space1 = new Canvas();
```

```
    space2 = new Canvas();
    space3 = new Canvas();
    space4 = new Canvas();
    space5 = new Canvas();
    space6 = new Canvas();
    space7 = new Canvas();
    space8 = new Canvas();
    space9 = new Canvas();
    space10 = new Canvas();

    northPanel = new Panel();
    northPanel.setLayout (new GridLayout(1,2));
    Button b1 = new Button("on");
    b1.addActionListener (this);
    b1.setActionCommand("b1");
    northPanel.add(b1);
    Button b2 = new Button("off");
    b2.addActionListener (this);
    b2.setActionCommand("b2");
    northPanel.add(b2);
    add(northPanel);
    add(space1);
    ServerName = new TextField("", 20);
    ServerName.setBackground(Color.gray);
    add(ServerName);
    add(space2);
    Port = new TextField("", 4);
    Port.setBackground(Color.gray);
    add(Port);
    add(space3);
    RefreshRatio = new TextField("", 4);
    RefreshRatio.setBackground(Color.gray);
    add(RefreshRatio);

    setVisible(true);
  }

//  ********************************************************************
//  *                                                                  *
//  *   public void actionPerformed( ActionEvent e)                    *
//  *                                                                  *
//  *   Description1:  sets values of control.unit, control.command, *
//  *                  control.data1 and control.data2 to certain   *
//  *                  values.                                       *
//  *                                                                  *
//  ********************************************************************

  public void actionPerformed (ActionEvent e) {
    String s = e.getActionCommand();
    if (s.equals("b1")) {
      control.command = "connect";
      control.unit = ServerName.getText();
      control.data1 = Integer.parseInt(Port.getText());
      control.data2 = Integer.parseInt(RefreshRatio.getText());
14:12 23.03.99    }
    else
    if (s.equals("b2")) {
      control.unit = "";
      control.command = "disconnect";
      control.unit = "";
      control.data1 = 0;
      control.data2 = 0;
    }

//  ********************************************************************
```

```
//  *                                                              *
//  *   Description2:  gives control of class protocoll of Applet      *
//  *                  Display_2 the same value as the contol variables*
//  *                  are in Connect                              *
//  *                                                              *
//  ****************************************************************

    resultApplet.control.command = control.command;
    resultApplet.control.unit = control.unit;
    resultApplet.control.data1 = control.data1;
    resultApplet.control.data2 = control.data2;
  }
}
```

Listings HTML

```
<HTML>

<HEAD><TITLE>P-NET Lab</TITLE></HEAD>

<BODY BGCOLOR="#ffffff" LINK="00006b" VLINK="56089f" ALINK="00006b">


<IMG SRC="images/masterhead_product.gif" WIDTH=600 HEIGHT=49 HSPACE=0 VSPACE=0
BORDER=0 ALIGN="top" ALT="IBM Partners"><BR CLEAR="all">

<TABLE WIDTH=600 HEIGHT=17 BORDER=0 CELLPADDING=0 CELLSPACING=0 VALIGN=top>

<TR HEIGHT=17><TD COLSPAN=2>
<A HREF="/"><IMG SRC="images/h_b.gif" BORDER=0 ALT="[ Home ]" ALIGN=LEFT
WIDTH=78 HEIGHT=17 HSPACE=0 VSPACE=0 HSPACE=0 VSPACE=0></A>
<A HREF="info.html"><IMG SRC="images/i_b.gif" BORDER=0 ALT="[ News ]"
ALIGN=LEFT WIDTH=77 HEIGHT=17 HSPACE=0 VSPACE=0></A>
<IMG SRC="images/p_h.gif" BORDER=0 ALT="[ Products &amp; services]" ALIGN=LEFT
WIDTH=129 HEIGHT=17 HSPACE=0 VSPACE=0>
<A HREF="cam.html"><IMG SRC="images/c_b.gif" BORDER=0 ALT="[ Webcam ]"
ALIGN=LEFT WIDTH=79 HEIGHT=17 HSPACE=0 VSPACE=0></A>
<IMG SRC="images/e_b.gif" BORDER=0 ALT="[ empty ]" ALIGN=LEFT WIDTH=79
HEIGHT=17 HSPACE=0 VSPACE=0>
<IMG SRC="images/e_b.gif" BORDER=0 ALT="[ empty ]" ALIGN=LEFT WIDTH=79
HEIGHT=17 HSPACE=0 VSPACE=0>
<IMG SRC="images/a_b.gif" BORDER=0 ALT="[ empty ]" ALIGN=LEFT WIDTH=79
HEIGHT=17 HSPACE=0 VSPACE=0>
</TABLE>

<TABLE WIDTH=600 BORDER=0 CELLPADDING=0 CELLSPACING=0 ALIGN="left">

<TR>



<TD WIDTH=468 VALIGN="TOP"><!--********* MAIN CONTENT **********-->
<TABLE align=left border=0 cellPadding=0 cellSpacing=0 width=468>

<TR>
<TD valign=top align=left bgColor=#003399 width=468 height=50 colspan=5><IMG
SRC="images/main_header.gif" ALT="" WIDTH=468 HEIGHT=50 HSPACE=0
BORDER=0></TD></TR>

<TR>
<TD valign=top align=left bgColor=#003399 width=36 height=408 ><IMG
SRC="images/main_fillerleft.gif"  WIDTH=36 HEIGHT=500 HSPACE=0 BORDER=0
ALT=""></TD>
<TD valign=top align=left bgColor=#003399 width=370 height=540>
```

```
<applet code="Display_2.class" name="Display" width=370 height=540> Sorry, no
display without Java.</applet></TD>
<TD valign=top align=left bgColor=#003399 width=62 height=500 rowSpan=4><IMG
SRC="images/main_fillerright.gif"  WIDTH=62 HEIGHT=500 HSPACE=0 BORDER=0
ALT=""></TD></TR>

<TR>
<TD valign=top align=left bgColor=#003399 width=468 colSpan=5 height=47><IMG
SRC="images/main_fillerbottom.gif" WIDTH=468 HEIGHT=47 HSPACE=0 BORDER=0
ALT=""></TD></TR>
<TR>
<TD valign=top align=left bgcolor=white width=468 height=17 colspan=5><IMG
SRC="images/main_footer.gif" WIDTH=468 HEIGHT=17 HSPACE=0 BORDER=0
ALT=""></TD></TR></TABLE>
<!--********* END MAIN CONTENT **********--><!--**Don't move***--></TD><!--
****Don't move***-->

<TD WIDTH=132 VALIGN="TOP">
<IMG SRC="images/left_menu_1.gif" WIDTH=132 HEIGHT=41 HSPACE=0 VSPACE=0
BORDER=0 ALT=""><IMG SRC="images/left_menu_2_l.gif" WIDTH=63 HEIGHT=272
HSPACE=0 VSPACE=0 BORDER=0 ALIGN="top" alt=""><applet code="ButtonTest.class"
name="Control1" width=57 height=272> Sorry, no display without
Java.</applet><IMG SRC="images/left_menu_2_r.gif" WIDTH=12 HEIGHT=272 HSPACE=0
VSPACE=0 BORDER=0 ALIGN="top" alt=""><IMG SRC="images/left_menu_4.gif"
WIDTH=132 HEIGHT=75 HSPACE=0 VSPACE=0 BORDER=0 ALIGN="top" ALT="">
<IMG SRC="images/left_menu_5_l.gif" WIDTH=63 HEIGHT=170 HSPACE=0 VSPACE=0
BORDER=0 ALIGN="top" ALT=""><applet code="Connect.class" name="Control2"
width=57 height=170> Sorry, no display without Java.</applet><IMG
SRC="images/left_menu_2_r.gif" WIDTH=12 HEIGHT=170 HSPACE=0 VSPACE=0 BORDER=0
ALIGN="top" ALT=""><IMG SRC="images/left_menu_water.gif" WIDTH=132 HEIGHT=51
HSPACE=0 VSPACE=0 BORDER=0 ALIGN="top" ALT=""><IMG
SRC="images/left_menu_3.gif" WIDTH=132 HEIGHT=28 HSPACE=0 VSPACE=0 BORDER=0
ALIGN="top" ALT="">
<A HREF="mailto:+436641431937@text.mobilkom.at"><IMG SRC="images/sms.gif"
WIDTH=41 HEIGHT=17 ALT="sms" HSPACE=0 VSPACE=0 BORDER=0 ALIGN="top"></A><A
HREF="mailto:e9326001@stud1.tuwien.ac.at"><IMG SRC="images/mail.gif" WIDTH=91
HEIGHT=17 ALT="mail" HSPACE=0 VSPACE=0 BORDER=0 ALIGN="top"></A><!--*****END
SEARCH CONSOLE*****--></TD>
</TD>

</TR>
</TABLE>
</BODY>
</HTML>
```

Listing Rost1:

```
{ Simple demo application for PD4000 }
{ RoSt2 R.STeiner & S.Hutter, 1999 }
PROGRAM SkeletonPD4000 [ objecttype=4000; capabilities= nobitaddress];

(*$I'..\..\Inc.4000\PD4000.SYS'*)                 (* System data declarations
*)
(*$I'..\..\PdLib\CH6x7.CHR'*)                     (* Character generator 6x7
*)
(*$I'..\..\PdLib\PDMODULE.DEF'*)                   (* Interface declarations
*)
(*$I'..\..\Inc.4000\PD4000SP.INC'*)
          (* This file holds various standard procedures to set up the PD4000
*)
{-------------------------------------------------------------------------------
-}
{  Your first global data declarations.
}
```

```
{-------------------------------------------------------------------------
-}
CONST

TYPE

VAR
 UPI : PD3221 AT NET: (1,21) NAME: 'UPI Module';
 TemperCh -> UPI.Analog_In_1;
 Temper -> TemperCh.AnalogIn;

 WeightMod : PD3230 at net:(1,30) NAME: 'Weight Transmitter';
 Weight -> WeightMod.weight.weight1;

 Digi : PD3120 AT NET: (1,5,2,20) NAME: 'Digital I/O';
 Light -> Digi.Digital_IO_9.FlagReg[7];


{-------------------------------------------------------------------------
-}

(*$I'FKEY4000.INC'*)
          (* This file holds the program for the function keys. The file can
be
             changed depending on the application *)

(*$I'..\..\Inc.4000\Init4000.INC'*)
          (* This task has to be the very first task in your program, because
it
             takes care of the initialisation of the controller and display.
*)

(*$I'..\..\Inc.4000\Key4000.INC'*)         (* Keyboard Task, key management *)
(*$I'..\..\PdLib\InterErr.INC'*)           (* Transmission error handler task
*)


{-------------------------------------------------------------------------
-}
{
}
{  "Would you tell me, please, which way I ought to go from here"?
}
{
}
{  "That depends a good deal on where you want to go to," said the Cat.
}
{
}
{  Alice's Adventures in Wonderland
}
{  Lewis Carroll
}
{
}
{-------------------------------------------------------------------------
-}

TASK YourFirstTask;
CONST
  Line1 = 0;
  Line2 = 7;
  Line3 = 14;
```

```
BEGIN (* YourFirstTask *)
  PenTo(1,Line1);
  Display('Simple demo application');
  MoveCursor(0,6);  (* Move cursor one line down from upper left corner *)

  LOOP              (* Loop for ever *)
    PenTo(31,Line2);
    Display('Temperature: ');
    Display(Temper:4:1);
    PenTo(31,Line3);
    Display('Weight: ');
    Display(Weight:4:1);
    ChangeTask;
  END;(* Loop *)
END;(* YourFirstTask *)
{------------------------------------------------------------------------------
-}
END.(* SkeletonPD4000 *)
{==============================================================================
=}

{==============================================================================
=}
{
}
{  ProjectName  : PD4000
}
{  Unity        : INCLUDE
}
{  Name og date : FKEY4000.INC 99
}
{  Writer       :
}
{  Modification : R.Steiner und S.Hutter
}
{
}
{==============================================================================
=}
{
}
{==============================================================================
=}

{------------------------------------------------------------------------------
-}
{
}
{  ProjectName  : PD4000
}
{  Unity        : PROCEDURE
}
{  Name og date : FuncKeys 04-02-98
}
{  Writer       :
}
{  Modification : John Rene Christensen
}
{
}
{------------------------------------------------------------------------------
-}
{ Standard function key procedure for PD4000
}
```

```
{-------------------------------------------------------------------------
-}

PROCEDURE FuncKeys(NewKey: BYTE);

BEGIN
  CASE NewKey of
    01:Begin
       if (Light1 = true) then Begin
                                  if (light=true) then
                                     begin Light1:=false;
                                           Light2:=true;
                                     end
                                     else
                                     begin Light1:=false;
                                           Light2:=false;
                                     end
                                   end
                               else
                                  Begin
                                  if (Light=true) then
                                    Begin
                                     Light1:=true;
                                     Light2:=false;
                                    end
                                    else
                                      Begin
                                     Light1:=false;
                                     Light2:=false;
                                    end
                                    end
        End
    02: if (Light = false) then light:=true else Light:=false;
    03: ;
    08: ;
    09: ;
    10: ;
    15: ;
    16: ;
    17: ;
    22: ;
    23: ;
    24: ;
  END; (* Case *)
END; (* FuncKeys *)
{-------------------------------------------------------------------------
-}
```

Listings RoSt 2:

```
{ Simple demo application for PD4000 }
{ RoSt 2, from S.Hutter & R.Stiner 1999 }
PROGRAM SkeletonPD4000 [ objecttype=4000; capabilities= nobitaddress];

(*$I'..\..\Inc.4000\PD4000.SYS'*)                 (* System data declarations
*)
(*$I'..\..\PdLib\CH6x7.CHR'*)                     (* Character generator 6x7
*)
(*$I'..\..\PdLib\PDMODULE.DEF'*)                   (* Interface declarations
*)
(*$I'..\..\Inc.4000\PD4000SP.INC'*)
          (* This file holds various standard procedures to set up the PD4000
*)
{-------------------------------------------------------------------------------
-}
{  Your first global data declarations.
}
{-------------------------------------------------------------------------------
-}
CONST

TYPE


VAR
 progstep: Integer;
 endtemp: Real;
 Modus,oldmodus: Integer;
 Light : Boolean;
 UPI : PD3221 AT NET: (1,21) NAME: 'UPI Module';
 Temp1 -> UPI.Analog_In_1.AnalogIn;
 Temp2 -> UPI.Analog_In_2.AnalogIn;
 WeightMod : PD3230 at net:(1,30) NAME: 'Weight Transmitter';
 Weight -> WeightMod.weight.weight1;

 Digi : PD3120 AT NET: (1,5,2,20) NAME: 'Digital I/O';
 Lampe1 -> Digi.Digital_IO_9.FlagReg[7];
 Lampe2 -> Digi.Digital_IO_A.FlagReg[7];
 Lampe3 -> Digi.Digital_IO_B.FlagReg[7];

 Pumpe -> WeightMod.Digital_IO_4.FlagReg[7];

 Ventil1 -> UPI.Digital_IO_1.FlagReg[7];
 Ventil2 -> WeightMod.Digital_IO_1.FlagReg[7];
 Ventil3 -> WeightMod.Digital_IO_2.FlagReg[7];
 Ventil4 -> WeightMod.Digital_IO_3.FlagReg[7];
 Ventil5 -> UPI.Digital_IO_2.FlagReg[7];

 EndSens1 -> UPI.Digital_IO_5.FlagReg[6];
 EndSens2 -> WeightMod.Digital_IO_5.FlagReg[6];

 Heizung -> UPI.Digital_IO_3.FlagReg[7];
 Mischer -> UPI.Digital_IO_4.FlagReg[7];



{-------------------------------------------------------------------------------
-}

(*$I'FKEY4000.INC'*)
          (* This file holds the program for the function keys. The file can
be
             changed depending on the application *)
```

```
(*$I'..\..\Inc.4000\Init4000.INC'*)
          (* This task has to be the very first task in your program, because
it
             takes care of the initialisation of the controller and display.
*)

(*$I'..\..\Inc.4000\Key4000.INC'*)        (* Keyboard Task, key management *)
(*$I'..\..\PdLib\InterErr.INC'*)          (* Transmission error handler task
*)


{-------------------------------------------------------------------------
-}
{
}
{  "Would you tell me, please, which way I ought to go from here"?
}
{
}
{  "That depends a good deal on where you want to go to," said the Cat.
}
{
}
{  Alice's Adventures in Wonderland
}
{  Lewis Carroll
}
{
}
{-------------------------------------------------------------------------
-}

TASK YourFirstTask;
CONST
  Line1 = 0;
  Line2 = 7;
  Line3 = 14;

procedure clrscr;
begin
 PenTo(1,Line1);
 Display('                        ');
 PenTo(1,Line2);
 Display('                        ');
 PenTo(1,Line3);
 Display('                        ');
end;

BEGIN (* YourFirstTask *)
  Lampe1:=false;
  lampe2:=false;
  Lampe3:=false;
  Pumpe :=false;
  Ventil1:=false;
  Ventil2:=false;
  Ventil3:=false;
  Ventil4:=false;
  Ventil5:=false;
  PenTo(1,Line1);
  modus:=1;
  oldmodus:=99;
  progstep:=0;

  LOOP             (* Loop for ever *)
```

```
 if (modus=3) then
  begin
   if oldmodus<>3 then clrscr;
   oldmodus:=3;
   PenTo(10,Line1);
   Display('Ts1O: ');
   Display(Temp1:4:1);
   Display('  Ts2U: ');
   Display(Temp2:4:1);
   PenTo(10,Line2);
   Display('Gew: ');
   Display(Weight:4:2);
   PenTo(10,Line3);
   Display('ES1O: ');
   Display(EndSens1:1);
   Display(' ES2U: ');
   Display(EndSens2:1);
  end; (* end modus 3*)

 if (modus=2) then
   begin
    if oldmodus<>2 then clrscr;
    if (progstep=0) then
    begin
          PenTo(10,Line1);
          Display('Press Key to start');
    end;
    oldmodus:=2;

    if (progstep=1) then (* init step 1 *)
    begin
    PenTo(10,Line1);
          Display('                  ');
        PenTo(10,Line1);
          Display('Tank 1 filling');
          ventil3:=true;
          pumpe:=true;
          Lampe1:=true;
        progstep:=progstep+1
    end;
    if (progstep=2) then (* execute step 1 *)
        if ((EndSens1=true) or (pumpe=false) or (Modus<>2)or (prg-
step<>2))then
        begin
          pumpe:=false;
          Lampe1:=false;
          ventil3:=false;
          if (progstep=2) then progstep:=3;
        end;
    if (progstep=3) then (* init step 2 *)
    begin
    Endtemp:=Temp1+3;
          clrscr;
          PenTo(10,Line1);
          Display('Tank 1 heating to ');
          Display(Endtemp:4:1);
          heizung:=true;
          Lampe2:=true;
    end;
    if (progstep=4) then (* exec. step 2 *)
    Begin
          if (progstep=4) then
        begin
        PenTo(10,Line2);
          Display('current Temp : ');
```

```
            Display(Temp1:4:1);
            end;
          if ((temp1>=Endtemp) or (Modus<>2)or (progstep<>4)) then
            begin
            heizung:=false;
                  Lampe2:=false;
            if progstep=4 then progstep:=5;
            end;
      end;
      if (progstep=5) then (* init step 3 *)
      Begin
            clrscr;
            PenTo(10,Line1);
            Display('Tank 2 unfilling ');
            PenTo(10,Line2);
            Display('                    ');
            Ventil5:=true;
          progstep:=progstep+1;
      end;
      if (progstep=6) then (*exec. step 3 *)
      begin
        if  ((weight<0.4) or (Modus<>2) or (progstep<>6)) then
        begin
         ventil5:=false;
             if progstep=6 then progstep:=7;
        end;
      end;
      if (progstep=7) then (*init step 4 *)
      begin
            clrscr;
            PenTo(10,Line1);
            Display('Tank 2 filling to 1.4');
            Ventil1:=true;
            Ventil2:=true;
          progstep:=progstep+1;
      end;
      if (progstep=8) then (*exec. step 4 *)
      begin
            if (progstep=8) then
          begin
          PenTo(10,Line2);
            Display('current weight : ');
            Display(weight:4:1);
            end;
          if ((weight>=1.4) or (Modus<>2)or (modus<>8)) then
          begin
            ventil2:=false;
          if progstep=8 then progstep:=9;
          end;
      end;
      if (progstep=9) then
      begin
            PenTo(10,Line1);
            Display('programm finished !');
      end;
     end; (* end modus 2*)

  if (modus=1) then
     begin
      if oldmodus<>1 then clrscr;
      oldmodus:=1;
      PenTo(10,Line1);
      Display('Pnet Aplication RoSt');
      PenTo(10,Line2);
      Display('1:info 2:auto 3:man');
```

```
    PenTo(10,Line3);
    Display('Press Key to change');
   end; (* end modus 1*)

 if (heizung=true) then Lampe3:=true;
    ChangeTask;

  END;(* EndLoop *)


END;(* EndYourFirstTask *)

{-------------------------------------------------------------------------------
-}
END.(* EndSkeletonPD4000 *)
{===============================================================================
=}

{===============================================================================
=}
{
}
{  ProjectName  : PD4000
}
{  Unity        : INCLUDE
}
{  Name og date : FKEY4000.INC 1999                                        }
{  Writer       :
}
{  Modification : R.Steiner & S.Hutter
}
{
}
{===============================================================================
=}
{
}
{===============================================================================
=}

{-------------------------------------------------------------------------------
-}
{
}
{  ProjectName  : PD4000
}
{  Unity        : PROCEDURE
}
{  Name og date : FuncKeys 04-02-98
}
{  Writer       :
}
{  Modification : John Rene Christensen
}
{
}
{-------------------------------------------------------------------------------
-}
{ Standard function key procedure for PD4000
}
{-------------------------------------------------------------------------------
-}
```

```
PROCEDURE FuncKeys(NewKey: BYTE);

BEGIN
  CASE NewKey of
    01: if (modus=3) then if (Pumpe=false) then Pumpe:=true else Pumpe:=false;
    02: if (modus=3) then if (Heizung=false) then Heizung:=true else Hei-
zung:=false;
    03: if (modus=3) then if (Mischer=false) then Mischer:=true else Mi-
scher:=false;
    08: if (modus=3) then if (Ventil1=false) then Ventil1:=true else Ven-
til1:=false;
    09: if (modus=3) then if (Ventil2=false) then Ventil2:=true else Ven-
til2:=false;
    10: if (modus=3) then if (Ventil3=false) then Ventil3:=true else Ven-
til3:=false;
    15: if (modus=3) then if (Ventil4=false) then Ventil4:=true else Ven-
til4:=false;
    16: if (modus=3) then if (Ventil5=false) then Ventil5:=true else Ven-
til5:=false;
    17: if (modus=3) then if (Lampe1=false) then Lampe1:=true else
Lampe1:=false;
    18: modus:=1;
    19: begin progstep:=1;modus:=2;end;
    20: modus:=3;

    22: modus:=1;
    23: begin progstep:=1;modus:=2;end;
    24: modus:=3;

  END; (* Case *)
END; (* FuncKeys *)
{-----------------------------------------------------------------------
---------------}
```

Listings VC++ Server:

Temeprature.cpp

```
// Temperature.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "Temperature.h"
#include "TemperatureDlg.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CTemperatureApp

BEGIN_MESSAGE_MAP(CTemperatureApp, CWinApp)
    //{{AFX_MSG_MAP(CTemperatureApp)
           // NOTE - the ClassWizard will add and remove mapping macros here.
           //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG
    ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTemperatureApp construction

CTemperatureApp::CTemperatureApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CTemperatureApp object

CTemperatureApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CTemperatureApp initialization

BOOL CTemperatureApp::InitInstance()
{
    // Initialize OLE libraries
    if (!AfxOleInit())
    {
           AfxMessageBox(IDP_OLE_INIT_FAILED);
           return FALSE;
    }


    // Standard initialization
    // If you are not using these features and wish to reduce the size
    //  of your final executable, you should remove from the following
    //  the specific initialization routines you do not need.

#ifdef _AFXDLL
```

```
    Enable3dControls();             // Call this when using MFC in a shared
DLL
#else
    Enable3dControlsStatic(); // Call this when linking to MFC statically
#endif

    // Parse the command line to see if launched as OLE server
    if (RunEmbedded() || RunAutomated())
    {
          // Register all OLE server (factories) as running.  This enables the
          //  OLE libraries to create objects from other applications.
          COleTemplateServer::RegisterAll();

          // Application was run with /Embedding or /Automation.  Don't show
the
          //  main window in this case.
          return TRUE;
    }

    // When a server application is launched stand-alone, it is a good idea
    //  to update the system registry in case it has been damaged.
    COleObjectFactory::UpdateRegistryAll();

    CTemperatureDlg dlg;
    m_pMainWnd = &dlg;
    int nResponse = dlg.DoModal();
    if (nResponse == IDOK)
    {
          // TODO: Place code here to handle when the dialog is
          //  dismissed with OK
    }
    else if (nResponse == IDCANCEL)
    {
          // TODO: Place code here to handle when the dialog is
          //  dismissed with Cancel
    }

    // Since the dialog has been closed, return FALSE so that we exit the
    //  application, rather than start the application's message pump.
    return FALSE;
}
```

TemperatureDlg.cpp

```
// TemperatureDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Temperature.h"
#include "TemperatureDlg.h"
#include "Winsock.h"
#include <afxsock.h>




#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
```

```
#endif

#define MyTimer 1
// Variables for the C-Java-Ole Server
SOCKET serversocket = INVALID_SOCKET;
SOCKET sock = INVALID_SOCKET;
int rc;
int anzb=0;
char* recBuff = new char[10];
BOOL conncted=FALSE;
/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    //{{AFX_MSG(CAboutDlg)
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)

    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
          // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTemperatureDlg dialog

CTemperatureDlg::CTemperatureDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CTemperatureDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CTemperatureDlg)
    m_temp1 = _T("");
    m_temp2 = _T("");
```

```
    m_host= _T("local host as Server");
    m_port= 4712;
    m_javas = _T("Java Server not connected");
    m_gewicht = _T("");
    m_licht1 = FALSE;
    m_licht2 = FALSE;
    m_licht3 = FALSE;
    m_licht4 = FALSE;
    m_licht5 = FALSE;
    m_licht6 = FALSE;
    m_licht7 = FALSE;
    m_licht8 = FALSE;
    m_schalt1 = FALSE;
    m_schalt2 = FALSE;
    m_schalt3 = FALSE;
    m_schalt4 = FALSE;
    m_schalt5 = FALSE;
    m_schalt6 = FALSE;
    m_schalt7 = FALSE;
    m_schalt8 = FALSE;
    m_vent1= FALSE;
    m_vent2= FALSE;
    m_vent3= FALSE;
    m_vent4= FALSE;
    m_vent5= FALSE;
    m_es1= FALSE;
    m_es2= FALSE;
    m_pumpe= FALSE;
    m_mischer=FALSE;
    m_errorstring = _T("");
    m_heizung=FALSE;
    //}}AFX_DATA_INIT


    // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);







}

void CTemperatureDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CTemperatureDlg)
    DDX_Text(pDX, IDC_temp1, m_temp1);
    DDX_Text(pDX, IDC_temp2, m_temp2);
    DDX_Text(pDX, IDC_gewicht, m_gewicht);
    DDX_Text(pDX, IDC_javaserver, m_javas);

    DDX_Text(pDX, IDC_host, m_host);
    DDX_Text(pDX, IDC_port, m_port);

    DDX_Check(pDX, IDC_l1, m_licht1);
    DDX_Check(pDX, IDC_l2, m_licht2);
    DDX_Check(pDX, IDC_l3, m_licht3);
    DDX_Check(pDX, IDC_l4, m_licht4);
    DDX_Check(pDX, IDC_l5, m_licht5);
    DDX_Check(pDX, IDC_l6, m_licht6);
    DDX_Check(pDX, IDC_l7, m_licht7);
```

```
    DDX_Check(pDX, IDC_l8, m_licht8);
    DDX_Check(pDX, IDC_s1, m_schalt1);
    DDX_Check(pDX, IDC_s2, m_schalt2);
    DDX_Check(pDX, IDC_s3, m_schalt3);
    DDX_Check(pDX, IDC_s4, m_schalt4);
    DDX_Check(pDX, IDC_s5, m_schalt5);
    DDX_Check(pDX, IDC_s6, m_schalt6);
    DDX_Check(pDX, IDC_v1, m_vent1);
    DDX_Check(pDX, IDC_v2, m_vent2);
    DDX_Check(pDX, IDC_v3, m_vent3);
    DDX_Check(pDX, IDC_v4, m_vent4);
    DDX_Check(pDX, IDC_v5, m_vent5);
    DDX_Check(pDX, IDC_Es1, m_es1);
    DDX_Check(pDX, IDC_Es2, m_es2);
    DDX_Check(pDX, IDC_pumpe, m_pumpe);
    DDX_Check(pDX, IDC_mischer, m_mischer);
    DDX_Check(pDX, IDC_heizung, m_heizung);
    DDX_Text(pDX, IDC_EDIT2, m_errorstring);
    //}}AFX_DATA_MAP


}

BEGIN_MESSAGE_MAP(CTemperatureDlg, CDialog)
    //{{AFX_MSG_MAP(CTemperatureDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_WM_TIMER()
    ON_WM_DESTROY()
    ON_BN_CLICKED(IDC_l1, Onl1)
    ON_BN_CLICKED(IDC_l2, Onl2)
    ON_BN_CLICKED(IDC_l3, Onl3)
    ON_BN_CLICKED(IDC_l4, Onl4)
    ON_BN_CLICKED(IDC_l5, Onl5)
    ON_BN_CLICKED(IDC_l6, Onl6)
    ON_BN_CLICKED(IDC_l7, Onl7)
    ON_BN_CLICKED(IDC_l8, Onl8)
    ON_BN_CLICKED(IDC_v1, Onv1)
    ON_BN_CLICKED(IDC_v2, Onv2)
    ON_BN_CLICKED(IDC_v3, Onv3)
    ON_BN_CLICKED(IDC_v4, Onv4)
    ON_BN_CLICKED(IDC_v5, Onv5)
    ON_EN_KILLFOCUS(IDC_host, OnKillfocushost)
    ON_EN_KILLFOCUS(IDC_port, OnKillfocusport)
    ON_BN_CLICKED(IDC_mischer, Onmischer)
    ON_BN_CLICKED(IDC_pumpe, Onpumpe)
    ON_BN_CLICKED(IDC_connect, Onconnect)
    ON_BN_CLICKED(IDC_heizung, Onheizung)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTemperatureDlg message handlers

BOOL CTemperatureDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);
```

```
    CMenu* pSysMenu = GetSystemMenu(FALSE);
    CString strAboutMenu;
    strAboutMenu.LoadString(IDS_ABOUTBOX);
    if (!strAboutMenu.IsEmpty())
    {
          pSysMenu->AppendMenu(MF_SEPARATOR);
          pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
    }

    // Set the icon for this dialog.  The framework does this automatically
    //  when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);               // Set big icon
    SetIcon(m_hIcon, FALSE);        // Set small icon

//  pvigoobj = new IVigoPro;

    vobj= new IVigoPro;

    idTimer=SetTimer(MyTimer, 100, NULL);

    return TRUE;  // return TRUE  unless you set the focus to a control
}

void CTemperatureDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
          CAboutDlg dlgAbout;
          dlgAbout.DoModal();
    }
    else
    {
          CDialog::OnSysCommand(nID, lParam);
    }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CTemperatureDlg::OnPaint()
{
    if (IsIconic())
    {
          CPaintDC dc(this); // device context for painting

          SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

          // Center icon in client rectangle
          int cxIcon = GetSystemMetrics(SM_CXICON);
          int cyIcon = GetSystemMetrics(SM_CYICON);
          CRect rect;
          GetClientRect(&rect);
          int x = (rect.Width() - cxIcon + 1) / 2;
          int y = (rect.Height() - cyIcon + 1) / 2;

          // Draw the icon
          dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
          CDialog::OnPaint();
    }
}
```

```
// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CTemperatureDlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}

void CTemperatureDlg::OnTimer(UINT nIDEvent)
{
    char str[10];
    float r;
    VARIANT result;

    /* /result=pvigoobj->GetValue();
    r=result.fltVal;
    sprintf(str, "%5.1f", r);
    m_temp1=str;
    */
if (conncted==TRUE)
{
    anzb = recv(sock,recBuff,7,0);
    if (anzb==SOCKET_ERROR)
    {
          m_javas = _T("Error recive data");

    }
    if (recBuff[0]=='V')
    { if  (recBuff[1]=='1')
       { if ((recBuff[2]=='s'))
                        {
                        if (m_vent1==FALSE) Onv1();
                        recBuff="V1set  \n";
                        anzb = send(sock,recBuff,8,0);

                        if (anzb!=8)
                        m_javas=_T("error send data");
                        recBuff="########";
                        }
            if (recBuff[2]=='r')
                        {if (m_vent1==TRUE) Onv1();
                        recBuff="V1reset\n";
                        anzb = send(sock,recBuff,8,0);

                        if (anzb!=8)
                        m_javas=_T("error send data");
                        recBuff="########";
                        }
          // Ventil 1 einlsesn
          vobj->CreateDispatch("Vigo.Pro");
          vobj->SetPhysId("UPI.Digital_IO_1.FlagReg[7]");
          result=vobj->GetValue();
          m_vent1=result.bool;
          vobj->ReleaseDispatch();
    }
     if  (recBuff[1]=='2')
       { if ((recBuff[2]=='s'))
                        {
                        if (m_vent2==FALSE) Onv2();
                        recBuff="V2set  \n";
                        anzb = send(sock,recBuff,8,0);
                        if (anzb!=8)
                        m_javas=_T("error send data");
                        recBuff="########";
                        }
            if (recBuff[2]=='r')
```

```
                    {
                    if (m_vent2==TRUE) Onv2();
                    recBuff="V2reset\n";
                    anzb = send(sock,recBuff,8,0);

                    if (anzb!=8)
                    m_javas=_T("error send data");
                    recBuff="########";
                    }
        // Ventil 2 einlsesn
        vobj->CreateDispatch("Vigo.Pro");
        vobj->SetPhysId("Waage.Digital_IO_1.FlagReg[7]");
        result=vobj->GetValue();
        m_vent2=result.bool;
        vobj->ReleaseDispatch();
     }
    if  (recBuff[1]=='3')
      { if ((recBuff[2]=='s'))
                    {if (m_vent3==FALSE) Onv3();
                    recBuff="V3set  \n";
                    anzb = send(sock,recBuff,8,0);

                    if (anzb!=8)
                    m_javas=_T("error send data");
                    recBuff="########";
                    }
         if (recBuff[2]=='r')
                    {
                    if (m_vent3==TRUE) Onv3();
                    recBuff="V3reset\n";
                    anzb = send(sock,recBuff,8,0);

                    if (anzb!=8)
                    m_javas=_T("error send data");
                    recBuff="########";
                    }
        // Ventil 3 einlsesn
        vobj->CreateDispatch("Vigo.Pro");
        vobj->SetPhysId("Waage.Digital_IO_2.FlagReg[7]");
        result=vobj->GetValue();
        m_vent3=result.bool;
        vobj->ReleaseDispatch();
     }
    if  (recBuff[1]=='4')
      { if ((recBuff[2]=='s'))
                    {
                    if (m_vent4==FALSE) Onv4();
                    recBuff="V4set  \n";
                    anzb = send(sock,recBuff,8,0);
                    if (anzb!=8)
                    m_javas=_T("error send data");
                    recBuff="########";
                    }

         if (recBuff[2]=='r')
                    {
                    if (m_vent4==TRUE) Onv4();
                    recBuff="V4reset\n";
                    anzb = send(sock,recBuff,8,0);

                    if (anzb!=8)
                    m_javas=_T("error send data");
                    recBuff="########";
                    }
   // Ventil 4 einlsesn
```

```
        vobj->CreateDispatch("Vigo.Pro");
        vobj->SetPhysId("Waage.Digital_IO_3.FlagReg[7]");
        result=vobj->GetValue();
        m_vent4=result.bool;
        vobj->ReleaseDispatch();
     }
    if  (recBuff[1]=='5')
      { if ((recBuff[2]=='s'))
                      {if (m_vent5==FALSE) Onv5();
                      recBuff="V5set  \n";
                      anzb = send(sock,recBuff,8,0);

                      if (anzb!=8)
                      m_javas=_T("error send data");
                      recBuff="########";
                      }
           if (recBuff[2]=='r')
                      {if (m_vent5==TRUE) Onv5();
                      recBuff="V5reset\n";
                      anzb = send(sock,recBuff,8,0);

                      if (anzb!=8)
                      m_javas=_T("error send data");
                      recBuff="########";
                }

   // Ventil 5 einlsesn
        vobj->CreateDispatch("Vigo.Pro");
        vobj->SetPhysId("UPI.Digital_IO_2.FlagReg[7]");
        result=vobj->GetValue();
        m_vent5=result.bool;
        vobj->ReleaseDispatch();
        UpdateData(FALSE);
     }
   }
   if (recBuff[0]=='P')
   {
        if ((recBuff[2]=='s'))
                      {if (m_pumpe==FALSE) Onpumpe();
                      recBuff="P1set  \n";
                      anzb = send(sock,recBuff,8,0);

                      if (anzb!=8)
                      m_javas=_T("error send data");
                      recBuff="########";
                      }
           if (recBuff[2]=='r')
                      {if (m_pumpe==TRUE) Onpumpe();
                      recBuff="P1reset\n";
                      anzb = send(sock,recBuff,8,0);

                      if (anzb!=8)
                      m_javas=_T("error send data");
                      recBuff="########";
           }
        // Pumpe einlesen
        vobj->CreateDispatch("Vigo.Pro");
        vobj->SetPhysId("Waage.Digital_IO_4.FlagReg[7]");
        result=vobj->GetValue();
        m_pumpe=result.bool;
        vobj->ReleaseDispatch();
   }
   if (recBuff[0]=='M')
   {
        if ((recBuff[2]=='s'))
```

```
                        {if (m_mischer==FALSE) Onmischer();
                        recBuff="M1set  \n";
                        anzb = send(sock,recBuff,8,0);

                        if (anzb!=8)
                        m_javas=_T("error send data");
                        recBuff="########";
                        }
             if (recBuff[2]=='r')
                        {if (m_mischer==TRUE) Onmischer();
                        recBuff="M1reset\n";
                        anzb = send(sock,recBuff,8,0);

                        if (anzb!=8)
                        m_javas=_T("error send data");
                        recBuff="########";
                        }
              // Mischer einlsen
            vobj->CreateDispatch("Vigo.Pro");
            vobj->SetPhysId("UPI.Digital_IO_4.FlagReg[7]");
            result=vobj->GetValue();
            m_mischer=result.bool;
            vobj->ReleaseDispatch();

    }
    if (recBuff[0]=='R')
    { if  (recBuff[1]=='T')
       { if (recBuff[2]=='1')
                 { recBuff="m_temp1";
                   anzb = send(sock,recBuff,7,0);
                   if (anzb!=7)
                   m_javas=_T("error send data");
                 }
         }
    }

}
else
{
    // Tempertaur 1 einlesen
    vobj->CreateDispatch("Vigo.Pro");
    vobj->SetPhysId("UPI.Analog_In_1.AnalogIn");
    result=vobj->GetValue();
    r=result.fltVal;
    sprintf(str, "%5.1f", r);
    m_temp1=str;
    vobj->ReleaseDispatch();
    // Tempertaur 2 einlesen
    vobj->CreateDispatch("Vigo.Pro");
    vobj->SetPhysId("UPI.Analog_In_2.AnalogIn");
    result=vobj->GetValue();
    r=result.fltVal;
    sprintf(str, "%5.1f", r);
    m_temp2=str;
    vobj->ReleaseDispatch();
    //Gewicht einlesen
    vobj->CreateDispatch("Vigo.Pro");
    vobj->SetPhysId("Waage.WEIGHT.WEIGHT1");
    result=vobj->GetValue();
    r=result.fltVal;
    sprintf(str, "%5.1f", r);
    m_gewicht=str;
    vobj->ReleaseDispatch();
    vobj->CreateDispatch("Vigo.Pro");
    vobj->SetPhysId("UPI.DIGITAL_IN_1.FLAGREG[6]");
```

```
result=vobj->GetValue();

m_es1=result.bool;
vobj->ReleaseDispatch();

// EndSensor 2 einlesen
vobj->CreateDispatch("Vigo.Pro");
vobj->SetPhysId("Waage.DIGITAL_IN_1.FLAGREG[6]");
result=vobj->GetValue();

m_es2=result.bool;
vobj->ReleaseDispatch();
// Heizung einlesen
vobj->CreateDispatch("Vigo.Pro");
vobj->SetPhysId("UPI.ANALOG_IN_1.ANALOGIN");
result=vobj->GetValue();
m_heizung=result.bool;
vobj->ReleaseDispatch();
// Mischer einlsen
vobj->CreateDispatch("Vigo.Pro");
vobj->SetPhysId("UPI.Digital_IO_4.FlagReg[7]");
result=vobj->GetValue();
m_mischer=result.bool;
vobj->ReleaseDispatch();
// Pumpe einlesen
vobj->CreateDispatch("Vigo.Pro");
vobj->SetPhysId("Waage.Digital_IO_4.FlagReg[7]");
result=vobj->GetValue();
m_pumpe=result.bool;
vobj->ReleaseDispatch();
// Schalter 1 einlesen
vobj->CreateDispatch("Vigo.Pro");
vobj->SetPhysId("Digi.Digital_IO_1.FlagReg[6]");
result=vobj->GetValue();
m_schalt1=result.bool;
vobj->ReleaseDispatch();
// Schalter 2 einlesen
vobj->CreateDispatch("Vigo.Pro");
vobj->SetPhysId("Digi.Digital_IO_2.FlagReg[6]");
result=vobj->GetValue();
m_schalt2=result.bool;
vobj->ReleaseDispatch();
// Schalter 3 einlesen
vobj->CreateDispatch("Vigo.Pro");
vobj->SetPhysId("Digi.Digital_IO_3.FlagReg[6]");
result=vobj->GetValue();
m_schalt3=result.bool;
vobj->ReleaseDispatch();
// Schalter 4 einlesen
vobj->CreateDispatch("Vigo.Pro");
vobj->SetPhysId("Digi.Digital_IO_4.FlagReg[6]");
result=vobj->GetValue();
m_schalt4=result.bool;
vobj->ReleaseDispatch();
// Schalter 5 einlesen
vobj->CreateDispatch("Vigo.Pro");
vobj->SetPhysId("Digi.Digital_IO_5.FlagReg[6]");
result=vobj->GetValue();
m_schalt5=result.bool;
vobj->ReleaseDispatch();
// Schalter 6 einlesen
vobj->CreateDispatch("Vigo.Pro");
vobj->SetPhysId("Digi.Digital_IO_6.FlagReg[6]");
result=vobj->GetValue();
m_schalt6=result.bool;
```

```
vobj->ReleaseDispatch();
// Licht 1 einlsen
vobj->CreateDispatch("Vigo.Pro");
vobj->SetPhysId("Digi.Digital_IO_9.FlagReg[7]");
result=vobj->GetValue();
m_licht1=result.bool;
vobj->ReleaseDispatch();
// Licht 2 einlsen
vobj->CreateDispatch("Vigo.Pro");
vobj->SetPhysId("Digi.Digital_IO_A.FlagReg[7]");
result=vobj->GetValue();
m_licht2=result.bool;
vobj->ReleaseDispatch();
// Licht 3 einlsen
vobj->CreateDispatch("Vigo.Pro");
vobj->SetPhysId("Digi.Digital_IO_B.FlagReg[7]");
result=vobj->GetValue();
m_licht3=result.bool;
vobj->ReleaseDispatch();
// Licht 4 einlsen
vobj->CreateDispatch("Vigo.Pro");
vobj->SetPhysId("Digi.Digital_IO_C.FlagReg[7]");
result=vobj->GetValue();
m_licht4=result.bool;
vobj->ReleaseDispatch();
// Licht 5 einlsen
vobj->CreateDispatch("Vigo.Pro");
vobj->SetPhysId("Digi.Digital_IO_D.FlagReg[7]");
result=vobj->GetValue();
m_licht5=result.bool;
vobj->ReleaseDispatch();
// Licht 6 einlsen
vobj->CreateDispatch("Vigo.Pro");
vobj->SetPhysId("Digi.Digital_IO_E.FlagReg[7]");
result=vobj->GetValue();
m_licht6=result.bool;
vobj->ReleaseDispatch();
// Licht 7 einlsen
vobj->CreateDispatch("Vigo.Pro");
vobj->SetPhysId("Digi.Digital_IO_F.FlagReg[7]");
result=vobj->GetValue();
m_licht7=result.bool;
vobj->ReleaseDispatch();
// Licht 8 einlsen
vobj->CreateDispatch("Vigo.Pro");
vobj->SetPhysId("Digi.Digital_IO_10.FlagReg[7]");
result=vobj->GetValue();
m_licht8=result.bool;
vobj->ReleaseDispatch();
// Ventil 1 einlsesn
vobj->CreateDispatch("Vigo.Pro");
vobj->SetPhysId("UPI.Digital_IO_1.FlagReg[7]");
result=vobj->GetValue();
m_vent1=result.bool;
vobj->ReleaseDispatch();
// Ventil 2 einlsesn
vobj->CreateDispatch("Vigo.Pro");
vobj->SetPhysId("Waage.Digital_IO_1.FlagReg[7]");
result=vobj->GetValue();
m_vent2=result.bool;
vobj->ReleaseDispatch();
// Ventil 3 einlsesn
vobj->CreateDispatch("Vigo.Pro");
vobj->SetPhysId("Waage.Digital_IO_2.FlagReg[7]");
result=vobj->GetValue();
```

```
    m_vent3=result.bool;
    vobj->ReleaseDispatch();
    // Ventil 4 einlsesn
    vobj->CreateDispatch("Vigo.Pro");
    vobj->SetPhysId("Waage.Digital_IO_3.FlagReg[7]");
    result=vobj->GetValue();
    m_vent4=result.bool;
    vobj->ReleaseDispatch();
    // Ventil 5 einlsesn
    vobj->CreateDispatch("Vigo.Pro");
    vobj->SetPhysId("UPI.Digital_IO_2.FlagReg[7]");
    result=vobj->GetValue();
    m_vent5=result.bool;
    vobj->ReleaseDispatch();
    UpdateData(FALSE);
    // Applets einlesen
}



    CDialog::OnTimer(nIDEvent);
}

void CTemperatureDlg::OnDestroy()
{
    CDialog::OnDestroy();
    KillTimer(idTimer);
    delete vobj;
}




void CTemperatureDlg::Onl1()
{
    // TODO: Add your control notification handler code here
    VARIANT val;
    vobj->CreateDispatch("Vigo.Pro");
    vobj->SetPhysId("Digi.Digital_IO_9.FlagReg[7]");
    if (m_licht1==FALSE) val.bool=TRUE;
          else val.bool=FALSE;
    val.vt=11;
    vobj->SetValue(val);
    vobj->ReleaseDispatch();
}

void CTemperatureDlg::Onl2()
{
    // TODO: Add your control notification handler code here
    VARIANT val;
    vobj->CreateDispatch("Vigo.Pro");
    vobj->SetPhysId("Digi.Digital_IO_A.FlagReg[7]");
    if (m_licht2==FALSE) val.bool=TRUE;
          else val.bool=FALSE;
    val.vt=11;
    vobj->SetValue(val);
    vobj->ReleaseDispatch();

}

void CTemperatureDlg::Onl3()
{
    // TODO: Add your control notification handler code here
    VARIANT val;
    vobj->CreateDispatch("Vigo.Pro");
```

```
    vobj->SetPhysId("Digi.Digital_IO_B.FlagReg[7]");
    if (m_licht3==FALSE) val.bool=TRUE;
          else val.bool=FALSE;
    val.vt=11;
    vobj->SetValue(val);
    vobj->ReleaseDispatch();

}

void CTemperatureDlg::Onl4()
{
    // TODO: Add your control notification handler code here
    VARIANT val;
    vobj->CreateDispatch("Vigo.Pro");
    vobj->SetPhysId("Digi.Digital_IO_C.FlagReg[7]");
    if (m_licht4==FALSE) val.bool=TRUE;
          else val.bool=FALSE;
    val.vt=11;
    vobj->SetValue(val);
    vobj->ReleaseDispatch();
}

void CTemperatureDlg::Onl5()
{
    // TODO: Add your control notification handler code here
    VARIANT val;
    vobj->CreateDispatch("Vigo.Pro");
    vobj->SetPhysId("Digi.Digital_IO_D.FlagReg[7]");
    if (m_licht5==FALSE) val.bool=TRUE;
          else val.bool=FALSE;
    val.vt=11;
    vobj->SetValue(val);
    vobj->ReleaseDispatch();
}

void CTemperatureDlg::Onl6()
{
    // TODO: Add your control notification handler code here
    VARIANT val;
    vobj->CreateDispatch("Vigo.Pro");
    vobj->SetPhysId("Digi.Digital_IO_E.FlagReg[7]");
    if (m_licht6==FALSE) val.bool=TRUE;
          else val.bool=FALSE;
    val.vt=11;
    vobj->SetValue(val);
    vobj->ReleaseDispatch();
}

void CTemperatureDlg::Onl7()
{
    // TODO: Add your control notification handler code here
    VARIANT val;
    vobj->CreateDispatch("Vigo.Pro");
    vobj->SetPhysId("Digi.Digital_IO_F.FlagReg[7]");
    if (m_licht7==FALSE) val.bool=TRUE;
          else val.bool=FALSE;
    val.vt=11;
    vobj->SetValue(val);
    vobj->ReleaseDispatch();
}

void CTemperatureDlg::Onl8()
{
    // TODO: Add your control notification handler code here
    VARIANT val;
```

```
    vobj->CreateDispatch("Vigo.Pro");
    vobj->SetPhysId("Digi.Digital_IO_10.FlagReg[7]");
    if (m_licht8==FALSE) val.bool=TRUE;
          else val.bool=FALSE;
    val.vt=11;
    vobj->SetValue(val);
    vobj->ReleaseDispatch();
}

void CTemperatureDlg::Onv1()
{
    // TODO: Add your control notification handler code here
    VARIANT val;
    vobj->CreateDispatch("Vigo.Pro");
    vobj->SetPhysId("UPI.Digital_IO_1.FlagReg[7]");
    if (m_vent1==FALSE) {val.bool=TRUE;}
          else {val.bool=FALSE;}
    val.vt=11;
    vobj->SetValue(val);
    vobj->ReleaseDispatch();
}

void CTemperatureDlg::Onv2()
{
    // TODO: Add your control notification handler code here
    VARIANT val;
    vobj->CreateDispatch("Vigo.Pro");
    vobj->SetPhysId("Waage.Digital_IO_1.FlagReg[7]");
    if (m_vent2==FALSE) val.bool=TRUE;
          else val.bool=FALSE;
    val.vt=11;
    vobj->SetValue(val);
    vobj->ReleaseDispatch();
}

void CTemperatureDlg::Onv3()
{
    // TODO: Add your control notification handler code here
    VARIANT val;
    vobj->CreateDispatch("Vigo.Pro");
    vobj->SetPhysId("Waage.Digital_IO_2.FlagReg[7]");
    if (m_vent3==FALSE) val.bool=TRUE;
          else val.bool=FALSE;
    val.vt=11;
    vobj->SetValue(val);
    vobj->ReleaseDispatch();


}

void CTemperatureDlg::Onv4()
{
    // TODO: Add your control notification handler code here

    VARIANT val;
    vobj->CreateDispatch("Vigo.Pro");
    vobj->SetPhysId("Waage.Digital_IO_3.FlagReg[7]");
    if (m_vent4==FALSE) val.bool=TRUE;
          else val.bool=FALSE;
    val.vt=11;
    vobj->SetValue(val);
    vobj->ReleaseDispatch();

}
```

```
void CTemperatureDlg::Onv5()
{
    // TODO: Add your control notification handler code here

    VARIANT val;
    vobj->CreateDispatch("Vigo.Pro");
    vobj->SetPhysId("UPI.Digital_IO_2.FlagReg[7]");

    if (m_vent5==FALSE) val.bool=TRUE;
          else val.bool=FALSE;
    val.vt=11;
    vobj->SetValue(val);
    m_errorstring=vobj->GetErrorString();
    vobj->ReleaseDispatch();

}


void CTemperatureDlg::OnKillfocushost()
{
    // TODO: Add your control notification handler code here
    m_host="hallo";
}

void CTemperatureDlg::OnKillfocusport()
{
    // TODO: Add your control notification handler code here

}

void CTemperatureDlg::Onmischer()
{
    // TODO: Add your control notification handler code here
    VARIANT val;
    vobj->CreateDispatch("Vigo.Pro");

    vobj->SetPhysId("UPI.Digital_IO_4.FlagReg[7]");

    if (m_mischer==FALSE) val.bool=TRUE;
          else val.bool=FALSE;
    val.vt=11;

    vobj->SetValue(val);
    vobj->ReleaseDispatch();

}

void CTemperatureDlg::Onpumpe()
{
    // TODO: Add your control notification handler code here
    VARIANT val;
    vobj->CreateDispatch("Vigo.Pro");
    vobj->SetPhysId("Waage.DIGITAL_IO_4.FLAGREG[7]");

    if (m_pumpe==FALSE) val.bool=TRUE;
          else val.bool=FALSE;
    val.vt=11;
    vobj->SetValue(val);
    m_errorstring=vobj->GetErrorString();
    vobj->ReleaseDispatch();
}

void CTemperatureDlg::Onconnect()
{
    // TODO: Add your control notification handler code here
```

```
    if (!AfxSocketInit())
    {
          m_javas = _T("Error Init AFX");
    }
    serversocket = socket(AF_INET, SOCK_STREAM, 0);
    if (serversocket== INVALID_SOCKET)
    {
          m_javas = _T("Error create Socket");
    }
    struct sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(m_port);
    serverAddr.sin_addr.s_addr=htonl(INADDR_ANY);
    memset(&(serverAddr.sin_zero),0,8);
    rc = bind(serversocket,(struct sockaddr*) &serverAddr, sizeof(struct
sockaddr));
    if (rc == -1)
    {
          m_javas = _T("Error bind Socket");
    }
    rc = listen (serversocket, 5);
    if (rc == -1)
    {
          m_javas = _T("Error lsiten Socket");
    }
    struct sockaddr_in clientaddr;
    int clientaddrsize=sizeof(struct sockaddr_in);
    m_javas = _T("wait for client");
    do
          sock = accept(serversocket,(struct sockaddr*) &clientaddr,
&clientaddrsize);
    while ((sock == INVALID_SOCKET) && (errno == 10004));
    m_javas = _T("got client");
    conncted=TRUE;

}

void CTemperatureDlg::Onheizung()
{
    // TODO: Add your control notification handler code here
    // Heizung wechseln
    VARIANT val;
    vobj->CreateDispatch("Vigo.Pro");
    vobj->SetPhysId("UPI.ANALOG_IN_1.ANALOGIN");
    if (m_heizung==FALSE) val.bool=FALSE;  // never set the heater
          else val.bool=FALSE;
    val.vt=11;
    vobj->SetValue(val);
    vobj->ReleaseDispatch();


}
```

VIGOOLE.cpp

```
// Machine generated IDispatch wrapper class(es) created with ClassWizard

#include "stdafx.h"
#include "vigoole.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif



/////////////////////////////////////////////////////////////////////////////
// IVigoPro properties

CString IVigoPro::GetPhysId()
{
    CString result;
    GetProperty(0x3, VT_BSTR, (void*)&result);
    return result;
}

void IVigoPro::SetPhysId(LPCTSTR propVal)
{
    SetProperty(0x3, VT_BSTR, propVal);
}

CString IVigoPro::GetSubPhysId()
{
    CString result;
    GetProperty(0x4, VT_BSTR, (void*)&result);
    return result;
}

void IVigoPro::SetSubPhysId(LPCTSTR propVal)
{
    SetProperty(0x4, VT_BSTR, propVal);
}

VARIANT IVigoPro::GetValue()
{
    VARIANT result;
    GetProperty(0x5, VT_VARIANT, (void*)&result);
    return result;
}

void IVigoPro::SetValue(const VARIANT& propVal)
{
    SetProperty(0x5, VT_VARIANT, &propVal);
}

VARIANT IVigoPro::GetInValue()
{
    VARIANT result;
    GetProperty(0x6, VT_VARIANT, (void*)&result);
    return result;
}

void IVigoPro::SetInValue(const VARIANT& propVal)
{
    SetProperty(0x6, VT_VARIANT, &propVal);
}

CString IVigoPro::GetErrorString()
{
    CString result;
    GetProperty(0x7, VT_BSTR, (void*)&result);
    return result;
}
```

```
void IVigoPro::SetErrorString(LPCTSTR propVal)
{
    SetProperty(0x7, VT_BSTR, propVal);
}

VARIANT IVigoPro::Get_Value()
{
    VARIANT result;
    GetProperty(0x0, VT_VARIANT, (void*)&result);
    return result;
}

void IVigoPro::Set_Value(const VARIANT& propVal)
{
    SetProperty(0x0, VT_VARIANT, &propVal);
}

short IVigoPro::GetFunctionNo()
{
    short result;
    GetProperty(0x8, VT_I2, (void*)&result);
    return result;
}

void IVigoPro::SetFunctionNo(short propVal)
{
    SetProperty(0x8, VT_I2, propVal);
}

short IVigoPro::GetIDCNo()
{
    short result;
    GetProperty(0x9, VT_I2, (void*)&result);
    return result;
}

void IVigoPro::SetIDCNo(short propVal)
{
    SetProperty(0x9, VT_I2, propVal);
}

long IVigoPro::GetOffset()
{
    long result;
    GetProperty(0xa, VT_I4, (void*)&result);
    return result;
}

void IVigoPro::SetOffset(long propVal)
{
    SetProperty(0xa, VT_I4, propVal);
}

short IVigoPro::GetObjectType()
{
    short result;
    GetProperty(0xb, VT_I2, (void*)&result);
    return result;
}

void IVigoPro::SetObjectType(short propVal)
{
    SetProperty(0xb, VT_I2, propVal);
}
```

```
short IVigoPro::GetDataType()
{
    short result;
    GetProperty(0xc, VT_I2, (void*)&result);
    return result;
}

void IVigoPro::SetDataType(short propVal)
{
    SetProperty(0xc, VT_I2, propVal);
}

CString IVigoPro::GetErrorDLL()
{
    CString result;
    GetProperty(0xd, VT_BSTR, (void*)&result);
    return result;
}

void IVigoPro::SetErrorDLL(LPCTSTR propVal)
{
    SetProperty(0xd, VT_BSTR, propVal);
}

BOOL IVigoPro::GetWriteOnly()
{
    BOOL result;
    GetProperty(0xe, VT_BOOL, (void*)&result);
    return result;
}

void IVigoPro::SetWriteOnly(BOOL propVal)
{
    SetProperty(0xe, VT_BOOL, propVal);
}

BOOL IVigoPro::GetReadOnly()
{
    BOOL result;
    GetProperty(0xf, VT_BOOL, (void*)&result);
    return result;
}

void IVigoPro::SetReadOnly(BOOL propVal)
{
    SetProperty(0xf, VT_BOOL, propVal);
}

float IVigoPro::GetProgress()
{
    float result;
    GetProperty(0x10, VT_R4, (void*)&result);
    return result;
}

void IVigoPro::SetProgress(float propVal)
{
    SetProperty(0x10, VT_R4, propVal);
}

short IVigoPro::GetProgramState()
{
    short result;
    GetProperty(0x11, VT_I2, (void*)&result);
```

```
    return result;
}

void IVigoPro::SetProgramState(short propVal)
{
    SetProperty(0x11, VT_I2, propVal);
}

CString IVigoPro::GetFileName()
{
    CString result;
    GetProperty(0x12, VT_BSTR, (void*)&result);
    return result;
}

void IVigoPro::SetFileName(LPCTSTR propVal)
{
    SetProperty(0x12, VT_BSTR, propVal);
}

CString IVigoPro::GetModelName()
{
    CString result;
    GetProperty(0x13, VT_BSTR, (void*)&result);
    return result;
}

void IVigoPro::SetModelName(LPCTSTR propVal)
{
    SetProperty(0x13, VT_BSTR, propVal);
}

CString IVigoPro::GetRevision()
{
    CString result;
    GetProperty(0x14, VT_BSTR, (void*)&result);
    return result;
}

void IVigoPro::SetRevision(LPCTSTR propVal)
{
    SetProperty(0x14, VT_BSTR, propVal);
}

CString IVigoPro::GetProgramName()
{
    CString result;
    GetProperty(0x15, VT_BSTR, (void*)&result);
    return result;
}

void IVigoPro::SetProgramName(LPCTSTR propVal)
{
    SetProperty(0x15, VT_BSTR, propVal);
}

CString IVigoPro::GetExeArgument()
{
    CString result;
    GetProperty(0x16, VT_BSTR, (void*)&result);
    return result;
}

void IVigoPro::SetExeArgument(LPCTSTR propVal)
{
```

```
    SetProperty(0x16, VT_BSTR, propVal);
}

long IVigoPro::GetSize()
{
    long result;
    GetProperty(0x17, VT_I4, (void*)&result);
    return result;
}

void IVigoPro::SetSize(long propVal)
{
    SetProperty(0x17, VT_I4, propVal);
}

short IVigoPro::GetMaxretry()
{
    short result;
    GetProperty(0x18, VT_I2, (void*)&result);
    return result;
}

void IVigoPro::SetMaxretry(short propVal)
{
    SetProperty(0x18, VT_I2, propVal);
}

short IVigoPro::GetBitNo()
{
    short result;
    GetProperty(0x19, VT_I2, (void*)&result);
    return result;
}

void IVigoPro::SetBitNo(short propVal)
{
    SetProperty(0x19, VT_I2, propVal);
}

BOOL IVigoPro::GetDataReady()
{
    BOOL result;
    GetProperty(0x1a, VT_BOOL, (void*)&result);
    return result;
}

void IVigoPro::SetDataReady(BOOL propVal)
{
    SetProperty(0x1a, VT_BOOL, propVal);
}

CString IVigoPro::GetNodeAddress()
{
    CString result;
    GetProperty(0x1b, VT_BSTR, (void*)&result);
    return result;
}

void IVigoPro::SetNodeAddress(LPCTSTR propVal)
{
    SetProperty(0x1b, VT_BSTR, propVal);
}

CString IVigoPro::GetNodeCapabilities()
{
```

```
    CString result;
    GetProperty(0x1c, VT_BSTR, (void*)&result);
    return result;
}

void IVigoPro::SetNodeCapabilities(LPCTSTR propVal)
{
    SetProperty(0x1c, VT_BSTR, propVal);
}

long IVigoPro::GetSubOffset()
{
    long result;
    GetProperty(0x1d, VT_I4, (void*)&result);
    return result;
}

void IVigoPro::SetSubOffset(long propVal)
{
    SetProperty(0x1d, VT_I4, propVal);
}

long IVigoPro::GetSubSize()
{
    long result;
    GetProperty(0x1e, VT_I4, (void*)&result);
    return result;
}

void IVigoPro::SetSubSize(long propVal)
{
    SetProperty(0x1e, VT_I4, propVal);
}

short IVigoPro::GetSubDataType()
{
    short result;
    GetProperty(0x1f, VT_I2, (void*)&result);
    return result;
}

void IVigoPro::SetSubDataType(short propVal)
{
    SetProperty(0x1f, VT_I2, propVal);
}

BOOL IVigoPro::GetPhysAddress()
{
    BOOL result;
    GetProperty(0x20, VT_BOOL, (void*)&result);
    return result;
}

void IVigoPro::SetPhysAddress(BOOL propVal)
{
    SetProperty(0x20, VT_BOOL, propVal);
}

BOOL IVigoPro::GetEnableExceptions()
{
    BOOL result;
    GetProperty(0x1, VT_BOOL, (void*)&result);
    return result;
}
```

```
void IVigoPro::SetEnableExceptions(BOOL propVal)
{
    SetProperty(0x1, VT_BOOL, propVal);
}

long IVigoPro::GetInternalAddress()
{
    long result;
    GetProperty(0x21, VT_I4, (void*)&result);
    return result;
}

void IVigoPro::SetInternalAddress(long propVal)
{
    SetProperty(0x21, VT_I4, propVal);
}

CString IVigoPro::GetVendor()
{
    CString result;
    GetProperty(0x22, VT_BSTR, (void*)&result);
    return result;
}

void IVigoPro::SetVendor(LPCTSTR propVal)
{
    SetProperty(0x22, VT_BSTR, propVal);
}

short IVigoPro::GetErrorCode()
{
    short result;
    GetProperty(0x23, VT_I2, (void*)&result);
    return result;
}

void IVigoPro::SetErrorCode(short propVal)
{
    SetProperty(0x23, VT_I2, propVal);
}

VARIANT IVigoPro::GetUserData()
{
    VARIANT result;
    GetProperty(0x2, VT_VARIANT, (void*)&result);
    return result;
}

void IVigoPro::SetUserData(const VARIANT& propVal)
{
    SetProperty(0x2, VT_VARIANT, &propVal);
}

BOOL IVigoPro::GetFileAccess()
{
    BOOL result;
    GetProperty(0x24, VT_BOOL, (void*)&result);
    return result;
}

void IVigoPro::SetFileAccess(BOOL propVal)
{
    SetProperty(0x24, VT_BOOL, propVal);
}
```

```
BOOL IVigoPro::GetInformationInErrorcode()
{
    BOOL result;
    GetProperty(0x25, VT_BOOL, (void*)&result);
    return result;
}

void IVigoPro::SetInformationInErrorcode(BOOL propVal)
{
    SetProperty(0x25, VT_BOOL, propVal);
}

/////////////////////////////////////////////////////////////////////////////
// IVigoPro operations

void IVigoPro::Start()
{
    InvokeHelper(0x26, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void IVigoPro::Stop()
{
    InvokeHelper(0x27, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void IVigoPro::Resume()
{
    InvokeHelper(0x28, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void IVigoPro::Reset()
{
    InvokeHelper(0x29, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void IVigoPro::Kill()
{
    InvokeHelper(0x2a, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void IVigoPro::SelectProgram()
{
    InvokeHelper(0x2b, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void IVigoPro::UnselectProgram()
{
    InvokeHelper(0x2c, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void IVigoPro::Download()
{
    InvokeHelper(0x2d, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void IVigoPro::DeleteDomain()
{
    InvokeHelper(0x2e, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void IVigoPro::Upload()
{
    InvokeHelper(0x2f, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}
```

```
void IVigoPro::DoRead()
{
    InvokeHelper(0x30, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void IVigoPro::DoWrite()
{
    InvokeHelper(0x31, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void IVigoPro::InitDownload()
{
    InvokeHelper(0x32, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void IVigoPro::InitUpload()
{
    InvokeHelper(0x34, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void IVigoPro::TerminateDownload()
{
    InvokeHelper(0x33, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void IVigoPro::TerminateUpload()
{
    InvokeHelper(0x35, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void IVigoPro::ExAnd(const VARIANT& Mask)
{
    static BYTE parms[] =
           VTS_VARIANT;
    InvokeHelper(0x36, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
            &Mask);
}

void IVigoPro::ExOr(const VARIANT& Mask)
{
    static BYTE parms[] =
           VTS_VARIANT;
    InvokeHelper(0x37, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
            &Mask);
}

VARIANT IVigoPro::TestAndSet(const VARIANT& Val)
{
    VARIANT result;
    static BYTE parms[] =
           VTS_VARIANT;
    InvokeHelper(0x38, DISPATCH_METHOD, VT_VARIANT, (void*)&result, parms,
           &Val);
    return result;
}

void IVigoPro::StopSequence()
{
InvokeHelper(0x39, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);}
```