Programmierpraktikum

DEVELOPING A
VIGO- JAVA INTERFACE

Technical report

Michael Krainz
MNr:9526174

Vienna University of Technology
Institute of Computer Technology

October 1999

Mail to:

Michael Krainz<e9526174@stud3.tuwien.ac.at>
Mikhail Gordeev<mischa@ict.tuwien.ac.at>

Contents

I. INTRODUCTION ... 3

1 P-NET AND VIGO DESCRIPTION ... 3
2 PROBLEM DESCRIPTION ... 3
3 FIRST STEPS ... 3
4 SOLUTION ... 4

II. MANUAL PART ONE (HOW TO USE THE INTERFACE) ... 4

5 THE DLL STRUCTURE OF THE INTERFACE ... 4
6 INSTALLATION ... 4
7 HOW TO USE THE INTERFACE ... 5
8 ERROR HANDLING ... 5

III. MANUAL PART TWO (EXACT DESCRIPTION OF THE INTERFACE) ... 6

9 THE FILE STRUCTURE OF THE INTERFACE ... 6
10 THE JAVA PART ... 6
11 THE C++ DLL ... 8
The separate native language source file (JavaCon.c), ... 8
The source file JavaConImp.c ... 8
12 THE DELPHI PART ... 11

IV. CONCLUSION ... 12

V. APPENDIX ... 12

THE JAVACON.JAVA FILE ... 12
THE JAVACONIMP.C FILE (C++DLL) ... 13
THE JAVACON.C(C++DLL) ... 15
THE VIGO.DPR FILE(DELPHI DLL) ... 16
THE MAIN.JAVA FILE (EXAMPLE FOR A JAVA APPLICATION) ... 18

I. Introduction

1 P-NET and VIGO description

The P-NET Fieldbus is designed to connect distributed process components like process computers, intelligent sensors, actuators, I/O modules, field and central controllers, PLC' s etc. Also providing the configuration of nodes/sensors, data collection and down-loading of programs. Beneath many other advantages the remarkable features of the P-NET protocol are the low costs of installation and node implementations, the easy and little cost intensive expansion (costs rise linear with expansion) and the use of "intelligent P-NET modules", witch contain additional process orientated functions. So that many process steps could be processed within the module, this helps to keep the process data, that has to bee send over the Fieldbus, low.

To control a P-NET Fieldbus often a standard Windows PC is used. A PCI-card (the Fieldbus-controller) and a software called VIGO provide the connection to the Fieldbus. VIGO, developed by PROCES-DATA, is a Fieldbus Management System. It enables a physical plant to be described in terms of data, related data structures and where data is located. It also menages data security and integrity for data enquiries made within the plant. VIGO also has an OLE2 interface so that the functionality of VIGO can be used in other programmes or programming languages like Visual Basic, Delphi or Visual C++.
By the hand, OLE, developed by Microsoft, means Object Linking and Embedding, witch is an easy way to exchange data within different programs.

2 Problem description

The aim of my „Programmierpraktikum" was to develop a solution to access VIGO from Java. As I mentioned before, it is possible to access VIGO from Visual Basic, Delphi (a programming language based an Pascal) or from C++, but there was no special interface developed for Java.

The reason why Java was chosen, was on the one hand, that it is the programming language most independent from different operating systems , on the other hand it' s easy handling of ' network programming' . So with this interface it should be very easy to access a P- NET Fieldbus through any type of network, f. e. the Internet.

3 First steps

As mentioned before VIGO uses OLE2 objects to interact with other programs. So we have to find a way to create these objects in Java. Java, a Sun product, doesn't use this Microsoft standard. A programming language has to be found, that supports both, a connection to Java

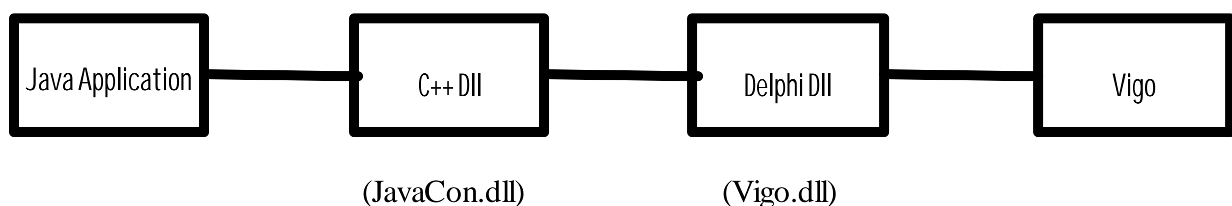
and OLE2 objects. C++ fulfils these requirements, it is possible to call C++ functions from Java with the **native methods** and it should be possible to create OLE2 objects. Thinking about the best way to call these C++ functions, I decided to store these functions in a C++ DLL. A DLL, Dynamic Link Library, is a collection of several functions in one file, with the suffix .dll. DLLs are a very common way to share several functions with different programs. Then I had to find out how to create C++ DLLs, how they could be used from a Java program, how to call them and how to pass and to receive data. As this worked properly I tried to implement the OLE2 part into the C++ DLL, but I had to find out that it didn' t work that properly, it supposed to. I downloaded the C++ example programs from Proces-data, but even these didn' t work properly on my computer. Although the size of the C++ DLL was very huge, when I implemented all the library files, needed for the OLE2 part. But the Delphi implementation worked properly and the size of the Delphi DLL file was very little.

4 Solution

I decided to use Delphi to access VIGO, with the OLE2 objects and to use C++ to establish the connection to Java. So I only had to find a way to make the C++ DLL communicate with the Delphi DLL. This wasn' t to difficult, because it' s something usual that two DLLs communicate with each other.

II. Manual part one (how to use the interface)

5 The Dll structure of the interface



(JavaCon.dll) (Vigo.dll)

I called the C++ Dll JavaCon.dll and the Delphi Dll Vigo.dll.
On the left side is the Java Application, which should communicate with the fieldbus. In this Application C++ methods will be called (this will be explained in the section "How to use the interface").These C++ methods, witch are stored in an DLL (dynamic link library) file, called *JavaCon.dll*, on their hands, call Delphi methods, which are stored in the DLL file *Vigo.dll* (described in *The VIGO part*).

6 Installation

The only thing to do is to copy the files (JavaCon.dll,Vigo.dll) into the directory of your Java Application on the PC, that is connected to the P- NET Fieldbus and make sure that VIGO is installed.

7 How to use the interface

To be able to call a method from *JavaCon.dll* you first have to create an Object of the JavaCon class to reference to:
JavaCon VigoObj; and
VigoObj = new JavaCon(); creates this Object.

By calling native methods you can access the fieldbus. You call these methods like any other, they only have to depend from the class *JavaCon*. In our case from the class variable *VigoObj,* like shown in the next example.

f. E. to call a method just use this Object
VigoObj.SetValue("UPI.ANALOG_IN_1.ANALOGIN",1);

the same with GetValue, f. E.:
private double Val;
Val:=VigoObj.GetValue("UPI.ANALOG_IN_1.ANALOGIN");
assigns the Value of UPI.ANALOG_IN_1.ANALOGIN to the Variable Val.

The main methods of *JavaCon.dll* are:
void SetValue(String,double),
double GetValue(String),
void SetBoolValue(String,boolean),
boolean GetBoolValue(String),
void closeCon().
With the use of SetValue you can set the value of a physical Id, specified by the String argument. GetValue returns a double with the value of the physical Id, specified by it' s String argument. The functions void SetBoolValue(String,boolean), boolean GetBoolValue(String) are used to pass or get a boolean value to the specified Object. The closeCon function has to be called to close the connection to the fieldbus.

With the statement *JavaCon VigoObj* we created a OLE object, so we **have to** destroy it before the program is closed. Otherwise an undefined error would occur, which determines the program. So *JavaCon.closeCon()* has to be called at the end of the program. To make sure that *closeCon()* is called even when an error occurred it is necessary to call the methods of the interface in a *try* and *catch* device, like in the example of the *Error handling* section. (I am still working on a solution that the closeCon() function is called automatically when the connection is closed)

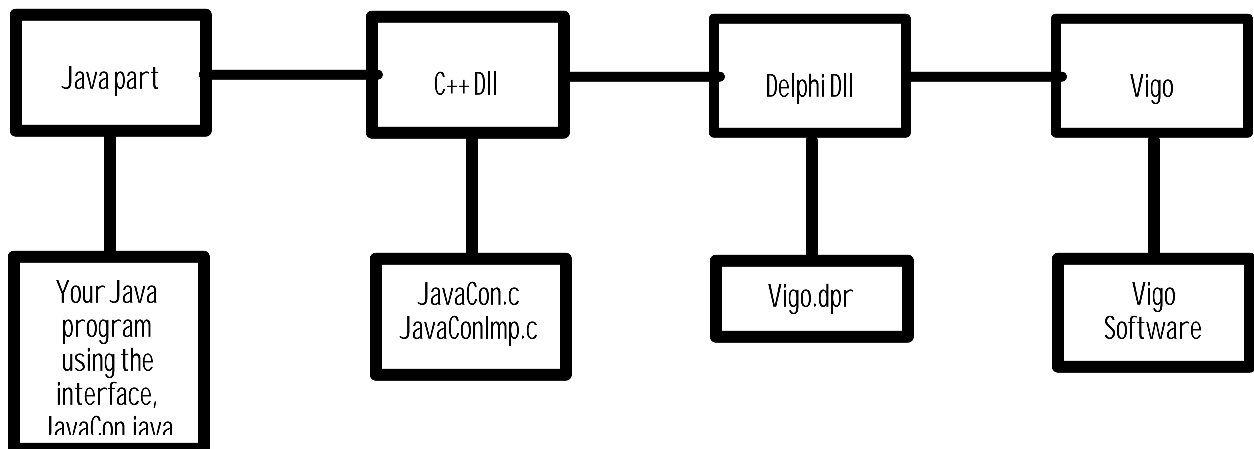
8 Error handling

Java' s error handling is fully implemented into the interface. So like any other Java method all methods of JavaCon will pass an ErrorString, if an error occurred, with the catch device.
For example: *try {*
VigoObj.SetValue("NOTEXISTINGNOD",1);
}catch(Exception E){System.out.println(E);}

would raise an exception, if the nod *"NOTEXISTINGNOD"* doesn' t exist.
The ErrorString, that is passed with the Error is in the errorhandler *"E"*.
The structure of "E" is the following: "*Exception:" ErrorString: ErrorNumber.*
Exception: only signifies that on Error occured and that it is from the instance *Error.*
ErrorString: passes the String that is produced by VIGO.
ErrorNumber: passes the in VIGO defined number of this error.

III. Manual Part two (Exact description of the interface)

9 The file structure of the interface



10 The Java part

Beginning with the left side of this graphic, the *Java part* includes your own Java program, that is using the interface (the chapter *how to use the interface* describes how to develop it) and the file *JavaCon.java*, which defines the native methods, that you use in your program. A native method has two components: the declaration and the implementation. You declare a native method, like other methods, in a Java class. You implement a native method in another programming language in a different source file. Typically, the implementation of a native method is a function. The first step is to create a class that defines the native methods. In our case it is in the JavaCon.java file.

```
class JavaCon
 {
 public native void test(int Val);   //testmethode für einfaches testen neuer Funktionen

1:    //Methodendeklarationen
   public native double GetValue(String ObjId) throws Exception;
      //liefert double-Wert eines P-NET Objektes
   public native boolean GetBoolValue(String ObjId) throws Exception;
      //liefert boolean-Wert eines P-NET Objektes
   public native void SetValue(String ObjId, double Val) throws Exception;
      ////setzt double-Wert eines P-NET Objektes
   public native void SetBoolValue(String ObjId, boolean Value) throws Exception;
      //setzt boolean-Wert eines P-NET Objektes
2:  public native void initCon();
      //Methode für die Initialisierung
    public native void closeCon();
      //Methode um Verbindung aubzubauen

3:  public JavaCon()      //wird automatisch bei erstem start von
    {                 //JavaCon.dll ausgeführt
    this.initCon();    //Initialisierungsmethode
    }
   static
    {
    try
      {
4:      System.loadLibrary("JavaCon");    //laden von JavaCon.dll
      } catch (UnsatisfiedLinkError e)
        {
        System.err.println("can't find JavaCon.dll");
        System.exit(-1);
        }
    }
   }
```

There are two sorts of native methods, declared in *JavaCon.java:*

2: First, *initCon()* and *closeCon()* are used to build up and to close the connection to the fieldbus.
3:*InitCon* is started automatically when an object of the class *JavaCon* is created. This is done by the statements:

public JavaCon()   //wird automatisch bei erstem Start von JavaCon.dll
// ausgeführt
{this.initCon();}   //Initialisierungsmethode}

CloseCon has to be called at the end of your program, using the interface, otherwise an undefined error would occur.

1: Second, the other methods, who actually access any node on the fieldbus. They are very similar; they only distinguish in their parameters.
For example:
public native double GetValue(String ObjId) throws Exeption;
This method definition provide only the method signature for *GetValue.* The first two words declare this method to a public native method, the third one, that it returns a double

value and the content of the brackets, that it passes a String argument. The *throws Exception* statement indicates that this method throws an Exception.

The implementation for these methods is provided in a separate native language source file (*JavaCon.c*), but this is explained later in section *The C++ DLL*.
4: The following static code block from the *JavaCon* class loads the appropriate library, named *JavaCon*.
System.loadLibrary("JavaCon");

11 The C++ DLL

To generate the C++ DLL, named *JavaCon.dll* two files will be needed.

The separate native language source file (JavaCon.c),

mentioned in paragraph before, will be created automaticaly with the *javah* utility and the *JavaConImp.c* file, that will be described later.
After compiling *JavaCon.java* with the command
In dos-mode: *javac JavaCon.java* use
In dos-mode: *javah JavaCon* to generate the *JavaCon.c* file.
This file is necessary for the C++DLL, because it defines the method names and their arguments, so that they can get accessed from the Java program, that is using the DLL.

The source file JavaConImp.c

The second file, we need for the C++DLL is *JavaConImp.c*, which contains the code of the methods, we have declared in *JavaCon.java*.
The full source code of this file is in the *Appendix*, here is a part of the listing, which describes the principle with an example of the function *JavaCon_GetValue*.

```
1:double JavaCon_GetValue(struct HJavaCon *this,Hjava_lang_String *ObjId)         //liefert double-Wert eines P-NET
Objektes
   {
    double Val=0;                              //Variable in der Wert gespeichert wird
    char buf[MaxLength];                                                          //char für Stringumwandlung
2:   javaString2CString(ObjId, buf, sizeof(buf));                                 //um JavaString in C++pchar
umzuwandeln
3:  Val=dlGetValue(buf);                                        //Wertzuweisung       von P-NET Objekt an Variable
4:  ErrorRep();                                                                                  //Error
vorhanden?
    return Val;                                                                          //Rückgabewert an Java
  }
```

1: The method name *JavaCon_GetValue* has to be used, because the automatically generated file *JavaCon.c* file uses this way to name the methods. The first part before the underscore defines the class that this method refers to, the second one the name of the method. The first argument passed with this method *struct HJavaCon *this* is the object

handler, the second one, *Hjava_lang_String *ObjId,* is the parameter we wanted to pass from Java side.
2: It is from the type of Hjava_lang_String, so we have to convert it into a C++ String. This is done with the statement *javaString2CString(ObjId, buf, sizeof(buf)).* The statement *Val=dlGetValue(buf);* receives the value from the node with the channelname of the String passed in *buf* . *dlGetValue* is the function of the Delphi DLL, *Vigo.dll*, that is described in the section *The Delphi DLL.*

4: The statement *ErrorRep();* calls the function that is used to check if an error occurred.

```
  void ErrorRep()                                   //Funktion zur Verwaltung von Errors
   {
    static CHAR szError[256];
1:   lstrcpyn((LPSTR)&szError,dlGetError(),sizeof(szError)); //kopieren des Rückgabewertes der dlGetError Funktion in
Vigo.dll
    //MessageBox(NULL, (LPSTR)&szError,NULL,MB_OK );
2:   if (szError[0])            //wenn ein Rückgabewert -> Error in Java-Teil erzeugen
          {
3:          SignalError(0, "java/lang/Exception",(LPSTR)&szError);
          }
   }
```

1: This method calls the function *dlGetError* of the Delphi DLL, which returns a String, and copies it' s value into the Array of Char *szError*.
2: If *szError* has a value, an error occurred and an error will be risen in the Java program, with the statement
3: *SignalError(0, "java/lang/Exception",(LPSTR)&szError).*
The first argument should be 0, the second describes the path of the error and the third passes an additional String, in our case the value returned from *dlGetError*.

Another very important part of *JavaConImp.c* is the part, where the connection to the Delphi Dll, *Vigo.dll*, is made. This is done in the *JavaCon_initCon()* method, which is called automatically when *JavaCon.dll* is accessed (addition information in the section *The Java part*). Again I only listed the part of *JavaConImp.c,* where this topic is treated with the example of the function *dlGetValue*, from the Delphi DLL. The listing beyond shows the necessary commands, to be able to call this method from C++. I wrote the line number before the statements, to make it easier to put the fragments together from the source code of the *Appendix* section.

```
13: static double (WINAPI FAR *dlGetValue) (String);
19: static HINSTANCE hinstVigo;
77:void JavaCon_initCon(struct HJavaCon *this)        //Initialisierungsmethode
   {
   hinstVigo = LoadLibrary("Vigo.DLL");//HINSTANCE-            Zuweisung und laden  von Vigo.dll

     if (HINSTANCE_ERROR > 23)                                    //Abfrage ob Vigo.dll geladen werden kann
          {    /* loaded successfully */
          (FARPROC) dlGetValue = GetProcAddress(hinstVigo, "dlGetValue");
102:        if (dlGetValue==NULL)                //Abfrage ob Methode vorhanden
             {
               MessageBox(NULL, "GetProcAddress failed of dlGetValue",NULL,MB_OK );
             return;
             }
          }
```

The first statement defines a pointer to a function called *dlGetValue* that returns a double and has one argument, a String.
The second statement,line 19, defines a hinstance object, that is used to identify the module (in our case the DLL *Vigo.dll*), that contains the functions, to be implemented.
The next statement from line 77 declares a function called *JavaCon_initCon(struct HJavaCon *this)*. We know this type of declaration from *The Java Dll* part. This function can be called from a Java program.
In the next line we load the library *Vigo.dll*. And the line after an if-query is started to see if an error occurred while loading this library.
In the next statement the pointer, we defined in the first statement gets the address of the module function's entry point.
This is done by the *GetProcAddress* function, which retrieves the address of the given module function.

FARPROC GetProcAddress(hinst, lpszProcName)

HINSTANCE hinst; / handle of module */*
LPCSTR lpszProcName; / address of function */*

Parameter Description

hinst Identifies the module that contains the function.
lpszProcName Points to a null-terminated string containing the function name, or specifies the ordinal value of the function.

Returns

The return value is the address of the module function's entry point if the GetProcAddress function is successful. Otherwise, it is NULL.

If the method defined by lpszProcName (in our case *dlGetValue*) doesn' t exist in this hinstance the return value will be null.
For this reason is the next statement in line 102; the if query.
The last statement only prints out if this method doesn' t exist.

Beneath the initialisation method we need the exit method too. From the *Java part* we know that it is called *closeCon()*, and so the name of this method in our C++DLL must be *JavaCon_closeCon()*.

```
void JavaCon_closeCon(struct HJavaCon *this)          //Methode um Verbindung aubzubauen
  {
  //MessageBox(NULL, "JavaCon_closeCon",NULL,MB_OK );
  dlExit();                               //Exit-Aufrunf in Vigo.dll
  FreeLibrary(hinstVigo);                          //Freigabe von Vigo.dll
  }
```

This method calls the Delphi method *dlExit()* from the Delphi DLL, *Vigo.dll* and closes this DLL afterwards(*FreeLibrary(hinstVigo)*).

12 The Delphi part

The Delphi DLL consists of one file called *Vigo.dpr*. I choose to call the methods of this DLL like the methods in *JacaCon.java*(*The Java Part*) except they are beginning with the letters “dl”(*delphi language*). So for example the method *GetValue* of *JavaCon.java* is called *dlGetValue* in the Delphi part.

```
1:function dlGetValue(ObjId:pchar):double;stdcall;     //liefert double-Wert eines P-NET Objektes
   begin
     //ShowMessage('dlGetValue: '+ObjId);
2:    dummy:=ObjId;                                     //Umwandlung von pchar in Sting
3:    dlObj.PhysID := dummy;                                    //Zuweisung der Id des P-NET Objektes(übergebener
Wert) an dlObj
4:    result:=dlObj.Value;                              //Rückgabe des Wertes
   end;
```

This method shows the principle that is used. From the method declaration we can see, that the argument, that is passed when the method is called, is a pchar and the return value is a double. I used the pchar type, because it is structured the same way a C++ String is structured. As we know from the section *The C++ DLL* we call the Delphi method *dlGetValue* and pass a C++ String as the argument.
With the second relevant command we change the pchar into a Delphi String.
Third we assign the Id of the VIGO object *dlObj* with the value of this String. The last relevant command returns the value (*result*), that we receive from the Fieldbus, of the VIGO object (*dlObj.Value*).

The VIGO object *dlObj* we used in the last paragraph has to be initialised. This happens in the function *dlInit*.

```
Procedure dlInit;stdcall;                                          //Initialisierungsmethode
begin
   inc(AcessCount);
     //Abfrage og erster Zugriff auf dll -> wenn ja,
   if AcessCount=1 then dlObj := CreateOleObject ('VIGO.STD');  //Erzeugung eines OLE-Objektes für Zugriff auf P-
NET                                                                //Objekte
end;
```

For reason of multi-user capability, the DLL has to ensure, that only one VIGO object (*dlObj*) is created. So even when more users access the Fieldbus at the same time, they share the same object to manipulate data on the Fieldbus, which is necessary for the stability of the interface and the data integrity. Every time a new user starts the interface the *AcessCount* will be incremented and only the first one generates the object (*dlObj:=CreateOleObject(‘VIGO.STD’)*).

So we have to close this object we created in *dlInit*, when we want to shut down the connection with the Fieldbus. This happens in the procedure *dlExit* with the statement *dlObj:=NULL*. But again we have to ensure that only the last user (if more users use the interface at the same time) closes this object. Again we use the variable *AcessCount*, that gets incremented by the number of users, who establish the connection(*dlInit*) and decrements by the number of users, who shut down the connection. Only if this variable is zero *dlObj* is destroyed.

```
procedure dlExit;stdcall;                       //Methode um Verbindung aubzubauen
begin
   dec(AcessCount);
     if AcessCount = 0 then dlObj := NULL;                //wenn letzter Zugreifer auf dll->löschen des OLE-Objektes
  end;
```

For the error handling, that is implemented, we also need a function, *dlGetError*.

```
function dlGetError:PChar;                              //Funktion zur Verwaltung von Errors
   var ErrorString:String;
       dummy:String;
       a:Integer;
   begin
1:    str(integer(dlObj.ErrorCode),dummy);                          //Umwandel des ErrorCode(num. Wert) in String
2:    ErrorString:=dlObj.ErrorString+' :'+dummy;        //ErrorString = Stringbeschreibung des Errors + ErrorCode
3:    for a:=1 to Length(ErrorString) Do                            //Umwandel des ErrorString in Pchar zur Übergabe
an C++
       begin
         szError[a-1]:=ErrorString[a];
       end;
     szError[Length(ErrorString)]:=#0;
5:   result:=addr(szError);                              //Rückgabe des pchar
   //ShowMessage(szError+'ist delphi Error');
  end;
```

This function returns a String, that consists of the error String and the error number, returned by VIGO.
In the first relevant line error number, an Integer, changes into a String, stored in *dummy*. In the second the variable *ErrorString* is put together, with the error String from VIGO (*dlObj.ErrorString*) and the error number (stored in *dummy*).
In the third relevant part, the *ErrorString*, a String variable is converted into a pchar to be able to return it to the C++DLL. The fifth statement actually returns this pchar.

IV. Conclusion

I hope, that I could manage to write a report that is easy understandable. To get more detailed information about **native methods(jni)** please refer to the web sites of javasoft.com. Additional information about the structure of DLLs can be found in the help files of your C++ and Delphi compiler. Thank You, Dear reader for your patience.

V. Appendix

The JavaCon.java file

```
class JavaCon
  {
  public native void test(int Val);   //testmethode für einfaches testen neuer Funktionen

      //Methodendeklarationen
```

```
public native double GetValue(String ObjId) throws Exception;
     //liefert double-Wert eines P-NET Objektes
public native boolean GetBoolValue(String ObjId) throws Exception;
     //liefert boolean-Wert eines P-NET Objektes
public native void SetValue(String ObjId, double Val) throws Exception;
     ////setzt double-Wert eines P-NET Objektes
public native void SetBoolValue(String ObjId, boolean Value) throws Exception;
     //setzt boolean-Wert eines P-NET Objektes
public native void initCon();
     //Methode für die Initialisierung
public native void closeCon();
     //Methode um Verbindung aubzubauen

public JavaCon()      //wird automatisch bei erstem start von
 {                 //JavaCon.dll ausgeführt
 this.initCon();     //Initialisierungsmethode
 }

static
 {
 try
   {
   System.loadLibrary("JavaCon");    //laden von JavaCon.dll
   } catch (UnsatisfiedLinkError e)
     {
     System.err.println("can't find JavaCon.dll");
     System.exit(-1);
     }
 }
}
```

The JavaConImp.c file (C++DLL)

```
#include <StubPreamble.h>
#include <javaString.h>
#include <stdio.h>
#include <windows.h>
#include <sys/types.h>
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
//Pointer auf Funktionen in Vigo.dll
static LONG (FAR *test) (LONG);
static void (WINAPI FAR *dlInit) ();
static void (WINAPI FAR *dlExit) ();
static double (WINAPI FAR *dlGetValue) (String);
static BOOL (WINAPI FAR *dlGetBoolValue) (String);
static LONG (WINAPI FAR *dlSetValue) (String, LONG);
static LONG (WINAPI FAR *dlSetBoolValue) (String, boolean);
static LPSTR (WINAPI FAR *dlGetError) ();

static HINSTANCE hinstVigo;
#define MaxLength 36  //max. Länge der zu übergebenden Strings

void JavaCon_test(struct HJavaCon *this,LONG Val)
 {
 test(3);
 SignalError(0, "java/lang/IllegalArgumentException","Erklärung");
 }

void ErrorRep()                                   //Funktion zur Verwaltung von Errors
 {
  static CHAR szError[256];
  lstrcpyn((LPSTR)&szError,dlGetError(),sizeof(szError)); //kopieren des Rückgabewertes der dlGetError Funktion in
Vigo.dll
  //MessageBox(NULL, (LPSTR)&szError,NULL,MB_OK );
  if (szError[0])              //wenn ein Rückgabewert -> Error in Java-Teil erzeugen
```

```
        {
                            SignalError(0, "java/lang/Exception",(LPSTR)&szError);
                            szError[0]="";
                           }
  }

double JavaCon_GetValue(struct HJavaCon *this,Hjava_lang_String *ObjId)          //liefert double-Wert eines P-NET
Objektes
 {
  double Val=0;                                                                  //Variable in der Wert gespeichert
wird
  char buf[MaxLength];                                                  //char für Stringumwandlung
  javaString2CString(ObjId, buf, sizeof(buf));            //um JavaString in C++pchar umzuwandeln
  //MessageBox(NULL, "JavaCon_GetValue",NULL,MB_OK );
  Val=dlGetValue(buf);                                                  //Wertzuweisung      von P-NET Objekt an
Variable
  ErrorRep();                                                           //Error vorhanden?
  return Val;                                                           //Rückgabewert an Java
 }
void JavaCon_SetValue(struct HJavaCon *this,Hjava_lang_String *ObjId,DOUBLE Value) //setzt double-Wert eines P-
NET
  {                                                                              //Objektes
  char buf[MaxLength];                                                           //char für Stringumwandlung
  javaString2CString(ObjId, buf, sizeof(buf));            //um JavaString in C++pchar umzuwandeln
  //MessageBox(NULL, "JavaCon_SetValue",NULL,MB_OK );
  dlSetValue(buf,Value);                                                //Wertzuweisung
  ErrorRep();                                                           //Error vorhanden?
  }

BOOL JavaCon_GetBoolValue(struct HJavaCon *this,Hjava_lang_String *ObjId)  //liefert boolean-Wert eines P-NET
Objektes
 {
  BOOL Val;                                                             //Variable in der Wert gespeichert wird
  char buf[MaxLength];                                                  //char für Stringumwandlung
  javaString2CString(ObjId, buf, sizeof(buf));            //um JavaString in C++pchar umzuwandeln
 //MessageBox(NULL, "JavaCon_GetValue",NULL,MB_OK );
  Val=dlGetBoolValue((LONG)&buf);                              //Wertzuweisung von P-NET Objekt an Variable
  ErrorRep();                                                  //Error vorhanden?
  return Val;                                                  //Rückgabewert an Java
 }
void JavaCon_SetBoolValue(struct HJavaCon *this,Hjava_lang_String *ObjId,boolean Value) //setzt boolean-Wert eines
P-NET
 {                                                                               //Objektes
  char buf[MaxLength];                                                           //char für Stringumwandlung
  javaString2CString(ObjId, buf, sizeof(buf));            //um JavaString in C++pchar umzuwandeln
  //MessageBox(NULL, "JavaCon_SetValue",NULL,MB_OK );
  dlSetBoolValue((LONG)&buf,Value);                            //Wertzuweisung
  ErrorRep();                                                  //Error vorhanden?
 }

void JavaCon_initCon(struct HJavaCon *this)             //Initialisierungsmethode
 {
 hinstVigo = LoadLibrary("Vigo.DLL");                          //HINSTANCE-Zuweisung und laden von Vigo.dll

    if (HINSTANCE_ERROR > 23)                                           //Abfrage ob Vigo.dll geladen werden kann
        {   /* loaded successfully */
               (FARPROC) test = GetProcAddress(hinstVigo, "dltest");
      if (test==NULL)                                          //Abfrage ob Methode vorhanden
          {
       MessageBox(NULL, "GetProcAddress failed of test",NULL,MB_OK );
                 return;
                 }
               (FARPROC) dlInit = GetProcAddress(hinstVigo, "dlInit");
      if (dlInit==NULL)                                        //Abfrage ob Methode vorhanden
          {
       MessageBox(NULL, "GetProcAddress failed of Init in Delphi dll",NULL,MB_OK );
                 return;
                 }
```

```
            (FARPROC) dlExit = GetProcAddress(hinstVigo, "dlExit");
      if (dlExit==NULL)                                    //Abfrage ob Methode vorhanden
        {
       MessageBox(NULL, "GetProcAddress failed of Exit in Delphi dll",NULL,MB_OK );
             return;
             }
            (FARPROC) dlGetValue = GetProcAddress(hinstVigo, "dlGetValue");
      if (dlGetValue==NULL)                                //Abfrage ob Methode vorhanden
        {
       MessageBox(NULL, "GetProcAddress failed of dlGetValue",NULL,MB_OK );
             return;
             }
            (FARPROC) dlSetValue = GetProcAddress(hinstVigo, "dlSetValue");
      if (dlSetValue==NULL)                                //Abfrage ob Methode vorhanden
        {
       MessageBox(NULL, "GetProcAddress failed of dlSetValue",NULL,MB_OK );
             return;
                 }
            (FARPROC) dlGetBoolValue = GetProcAddress(hinstVigo, "dlGetBoolValue");
      if (dlGetBoolValue==NULL)                            //Abfrage ob Methode vorhanden
        {
       MessageBox(NULL, "GetProcAddress failed of dlGetBoolValue",NULL,MB_OK );
             return;
             }
            (FARPROC) dlSetBoolValue = GetProcAddress(hinstVigo, "dlSetBoolValue");
      if (dlSetBoolValue==NULL)                   //Abfrage ob Methode vorhanden
        {
       MessageBox(NULL, "GetProcAddress failed of dlSetBoolValue",NULL,MB_OK );
             return;
                 }
            (FARPROC) dlGetError = GetProcAddress(hinstVigo, "dlGetError");
      if (dlGetError==NULL)                                //Abfrage ob Methode vorhanden
        {
       MessageBox(NULL, "GetProcAddress failed of dlGetError",NULL,MB_OK );
             return;
             }
            //MessageBox(NULL, "JavaCon_init",NULL,MB_OK );
            dlInit();                                          //Aufrunf der Initialisierungsmethode in
Vigo.dll

           }
           else {
        MessageBox(NULL, "LoadLibrary failed",NULL,MB_OK );
        return;
        }
    }
void JavaCon_closeCon(struct HJavaCon *this)         //Methode um Verbindung aufzubauen
  {
  //MessageBox(NULL, "JavaCon_closeCon",NULL,MB_OK );
  dlExit();                                              //Exit-Aufrunf in Vigo.dll
  FreeLibrary(hinstVigo);                                //Freigabe von Vigo.dll
  }
```

The JavaCon.c(C++DLL)

This file is generated automatically and is only listed to complete the picture.

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <StubPreamble.h>

/* Stubs for class JavaCon */
/* SYMBOL: "JavaCon/test(I)V", Java_JavaCon_test_stub */
__declspec(dllexport) stack_item *Java_JavaCon_test_stub(stack_item *_P_,struct execenv *_EE_) {
    extern void JavaCon_test(void *,long);
```

```
    (void) JavaCon_test(_P_[0].p,((_P_[1].i)));
    return _P_;
}
/* SYMBOL: "JavaCon/GetValue(Ljava/lang/String;)D", Java_JavaCon_GetValue_stub */
__declspec(dllexport) stack_item *Java_JavaCon_GetValue_stub(stack_item *_P_,struct execenv *_EE_) {
    Java8 _tval;
    extern double JavaCon_GetValue(void *,void *);
    SET_DOUBLE(_tval, _P_, JavaCon_GetValue(_P_[0].p,((_P_[1].p))));
    return _P_ + 2;
}
/* SYMBOL: "JavaCon/GetBoolValue(Ljava/lang/String;)Z", Java_JavaCon_GetBoolValue_stub */
__declspec(dllexport) stack_item *Java_JavaCon_GetBoolValue_stub(stack_item *_P_,struct execenv *_EE_) {
    extern long JavaCon_GetBoolValue(void *,void *);
    _P_[0].i = (JavaCon_GetBoolValue(_P_[0].p,((_P_[1].p))) ? TRUE : FALSE);
    return _P_ + 1;
}
/* SYMBOL: "JavaCon/SetValue(Ljava/lang/String;D)V", Java_JavaCon_SetValue_stub */
__declspec(dllexport) stack_item *Java_JavaCon_SetValue_stub(stack_item *_P_,struct execenv *_EE_) {
    Java8 _t2;
    extern void JavaCon_SetValue(void *,void *,double);
    (void) JavaCon_SetValue(_P_[0].p,((_P_[1].p)),GET_DOUBLE(_t2, _P_+2));
    return _P_;
}
/* SYMBOL: "JavaCon/SetBoolValue(Ljava/lang/String;Z)V", Java_JavaCon_SetBoolValue_stub */
__declspec(dllexport) stack_item *Java_JavaCon_SetBoolValue_stub(stack_item *_P_,struct execenv *_EE_) {
    extern void JavaCon_SetBoolValue(void *,void *,long);
    (void) JavaCon_SetBoolValue(_P_[0].p,((_P_[1].p)),((_P_[2].i)));
    return _P_;
}
/* SYMBOL: "JavaCon/initCon()V", Java_JavaCon_initCon_stub */
__declspec(dllexport) stack_item *Java_JavaCon_initCon_stub(stack_item *_P_,struct execenv *_EE_) {
    extern void JavaCon_initCon(void *);
    (void) JavaCon_initCon(_P_[0].p);
    return _P_;
}
/* SYMBOL: "JavaCon/closeCon()V", Java_JavaCon_closeCon_stub */
__declspec(dllexport) stack_item *Java_JavaCon_closeCon_stub(stack_item *_P_,struct execenv *_EE_) {
    extern void JavaCon_closeCon(void *);
    (void) JavaCon_closeCon(_P_[0].p);
    return _P_;
}
```

The Vigo.dpr file(Delphi DLL)

```
library vigo;

uses
  SysUtils,
  Classes,
  Dialogs,
  Windows,
  Messages,
  Graphics,
  Controls,
  Forms,
  OleAuto,
  ExtCtrls,
  StdCtrls;

var
  dlObj : Variant;              //Variant-Objekt mit späterer Zuweisung als OLE-Objekt
  dummy:String;
  szError:Array [0..255] of Char; //zur Speicherung des ErrorStrings
  SaveExit: Pointer;

Const
```

```
  AcessCount:LongInt=0;           //Zähler der Anzahl der Zugreifer auf dll

function dltest( A: LongInt):LongInt; {stdcall;} //testmethode
  begin
    ShowMessage('Anfang');
  end;
Procedure dlInit;stdcall;                        //Initialisierungsmethode

  begin
    inc(AcessCount);
     //Abfrage og erster Zugriff auf dll -> wenn ja,
    if AcessCount=1 then dlObj := CreateOleObject ('VIGO.STD');  //Erzeugung eines OLE-Objektes für Zugriff auf P-NET
                                                                                                          //Objekt
  end;
procedure dlExit;stdcall;                        //Methode um Verbindung aubzubauen
  Var sc_num:String;
  begin
    dec(AcessCount);
    Str(AcessCount,sc_num);
    if AcessCount = 0 then dlObj := NULL;        //wenn letzter Zugreifer auf dll->löschen des OLE-Objektes
    //ShowMessage('dlExit:'+sc_num);
  end;
function dlGetError:PChar;                       //Funktion zur Verwaltung von Errors
  var ErrorString:String;
      dummy:String;
      a:Integer;
  begin
    str(integer(dlObj.ErrorCode),dummy);          //Umwandel des ErrorCode(num. Wert) in String
    if (dummy = '0') then ErrorString:=dlObj.ErrorString
    else ErrorString:=dlObj.ErrorString+' :'+dummy;   //ErrorString = Stringbeschreibung des Errors + ErrorCode
    for a:=1 to Length(ErrorString) Do           //Umwandel des ErrorString in Pchar zur Übergabe an C++
      begin
        szError[a-1]:=ErrorString[a];
      end;
    szError[Length(ErrorString)]:=#0;
    result:=addr(szError);                       //Rückgabe des pchar
    //ShowMessage(szError+'ist delphi Error');
  end;
function dlGetValue(ObjId:pchar):double;stdcall;   //liefert double-Wert eines P-NET Objektes
  begin
    //ShowMessage('dlGetValue: '+ObjId);
    dummy:=ObjId;                                //Umwandlung von pchar in Sting
    dlObj.PhysID := dummy;                           //Zuweisung der Id des P-NET Objektes(übergebener Wert) an dlObj
    result:=dlObj.Value;                         //Rückgabe des Wertes
  end;
function dlSetValue(ObjId:pchar;dlValue:LongInt):LongInt;stdcall;   //setzt double-Wert eines P-NET Objektes
  begin
    //ShowMessage('dlSetValue: '+ObjId);
    dummy:=ObjId;                                //Umwandlung von pchar in Sting
    dlObj.PhysID := dummy;                           //Zuweisung der Id des P-NET Objektes(übergebener Wert) an dlObj
    dlObj.Value := dlValue;                       //setzen des übergebenen Wertes
  end;
function dlGetBoolValue(ObjId:pchar):bool;stdcall;  //liefert boolean-Wert eines P-NET Objektes
  begin
    //ShowMessage('dlGetBoolValue: '+ObjId);
    dummy:=ObjId;                                //Umwandlung von pchar in Sting
    dlObj.PhysID := dummy;                           //Zuweisung der Id des P-NET Objektes(übergebener Wert) an dlObj
    result:=dlObj.Value;                         //Rückgabe des Wertes
  end;
function dlSetBoolValue(ObjId:pchar;dlValue:bool):LongInt;stdcall;  //setzt boolean-Wert eines P-NET Objektes
  begin
    //ShowMessage('dlSetBoolValue: '+ObjId);
    dummy:=ObjId;                                //Umwandlung von pchar in Sting
    dlObj.PhysID := dummy;                           //Zuweisung der Id des P-NET Objektes(übergebener Wert) an dlObj
    dlObj.Value := dlValue;                       //setzen des übergebenen Wertes
  end;
exports
 //Mit Inex versehene Methoden, die zu exportieren sind
```

```
 dlInit index 1,dlExit index 2, dlGetValue index 3,
 dlSetValue index 4,dlGetBoolValue index 5,dlSetBoolValue index 6,dlGetError index 7, dltest index 8;


//Procedure zum schließen der dll
procedure LibExit;
  begin
    ExitProc := SaveExit;  { Kette der Exit-Prozeduren wiederherstellen }
  end;
begin
    AcessCount:=0;
 SaveExit := ExitProc;     { Kette der Exit-Prozeduren speichern }
 ExitProc := @LibExit;    { Exit-Prozedur LibExit installieren }
end.
```

The Main.java file (example for a Java application)

This example shows how the main functions could be used

```
class Main {
    public static void main(String[] args) {
    JavaCon VigoObj;
    VigoObj=new JavaCon();


    for (int i =0;i<11;i++)
      {
      try {
       VigoObj.SetValue("UPI.ANALOG_IN_1.ANALOGIN",i);
        }catch(Exception E){System.out.println(E);}
      }
    try {
      VigoObj.GetBoolValue("UPI.DIGITAL_IN_1.ANALOGIN");
      }catch(Exception E){System.out.println(E);}
    try {
      System.out.println(VigoObj.GetValue("UPI.ANALOG_IN_1.ANALOGIN"));

      }catch(Exception E){System.out.println(E);}

    VigoObj.closeCon();
    }
}
```