Interfacing with Power WinPLC

By Vanderhaegen Bart

1 Contents

This manuel is created by Vanderhaegen Bart

2 Introduction

Starting from version 6.1, WinPLC is capable to let other programs control certain functions. WinPLC offers a set of functions that can be called from other programs. The picture belows demonstrates this principe:

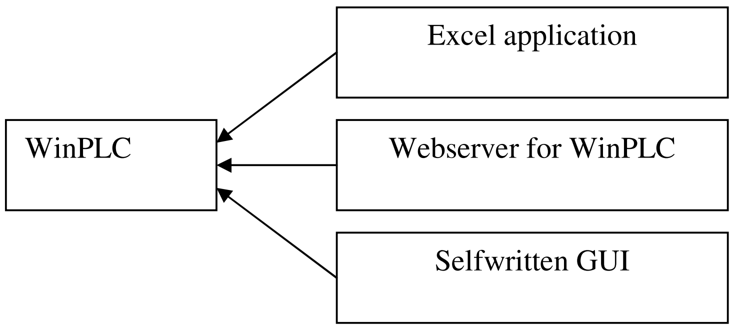


Figure 1: Some client programs are to WinPLC (acting as server)

In this image, WinPLC is showed as a server. There are 3 sample clients connected to the server:

- **Excel Application**. An excel worksheet[1] could be used to collect some data from the K8000 for historical graphs etc.
- **Webserver for WinPLC**. Someone could make a Webserver for WinPLC. When a remote user clicks a certain hyperlink, the webserver could activate I/O channels through WinPLC. Doing so, the webserver doesn't have to access the K8000 card directly.
- **Self-written GUI**. WinPLC allows the creation of a custom graphical user interface. This interface is limited. You could design your own graphical user interface in Visual Basic (or any other programming language), and connect the button-actions to WinPLC.

The clients can use WinPLC (and the K8000) at the same time, and can be made with a minimal amount of source code.

This manual contains examples and instructions to make other programs communicate with WinPLC. The samples can also be downloaded from my webpage: http://users.pandora.be/bartv/winplc/demos.zip

[1] VBA (Visual Basic for Applications) is a builtin language in Microsoft Excel. It allows you to program applications for your Excel worksheet. An example application would be a button, that retrieves information from the K8000 when the button is pressed.

The use of WinPLC as a server has many advantages:

- The controlling of the K8000 is centralized. Once WinPLC is setup to control the K8000 properly, other programs doesn’t have to worry connection settings.

- Multiple programs can control the K8000 simultane. Normally, the K8000 can only be controlled by one program at the same time. More programs will cause interference to each other.

- Coding for your custom applications can be minimized.

- The K8000 can be controlled from many programming languages, like C++, Visual Basic, Visual Basic for Applications, Windows Scripts, ASP (Active Server Pages) and even Java.

3 Available functions

WinPLC offers functions that can be divided in 3 groups.

3.1 Direct K8000 I/O Routines

The direct K8000 I/O Routines can control the I/O functions directly.

3.1.1 I/O Channels

- Public Sub setIOCh(Channel as Integer)

  This method will set an I/O channel. You need to give an integer as the parameter: a number between 1 and 64. (since there are 64 I/O channels, if you are using 4 K8000 cards).

- Public Sub clearIOCh(Channel as Integer)

  This method will clear an I/O channel. You need to give an integer as the parameter: a number between 1 and 64.

- Public Sub invertIOCh(Channel as Integer)

  This method will invert an I/O channel. You need to give an integer as the parameter: a number between 1 and 64.

- Public Sub setAllIOCh()

  This method will set all I/O channels (from 1 to 64).

- Public Sub clearAllIOCh()

  This method will clear all I/O channels (from 1 to 64)

- Public Sub invertAllIOCh()

  This method will invert all I/O channels (from 1 to 64)

- Public Function getIOState(channel as Integer)

  This function returns the state (1 or 0) of an I/O channel. (Channel is the channelnr between 1 and 64)

3.1.2 DAC Channels

- Public Sub setDACch(channel as Integer, value as Integer)

  This method will set a DAC channel to a certain value. You must specify the DAC channel, a number between 1 and 32, and the value you want to send to that DAC value: a number between 0 and 63.

- Public Sub setAllDACCh()

  This method will set all DAC channels to the maximal value (63).

- Public Sub clearAllDACCh()

  This method will clear all DAC channels to the minimal value (0).

- Public Sub getDACstate(channel as Integer)

  This function returns the value of a DAC channel. The channel must be specified, and is an integer between 1 and 32.

3.1.3 DA Channels

- Public Sub setDAch(channel as Integer, value as Integer)

  This method will set a DA channel to a certain value. You must specify the DA channel, a number between 1 and 4, and the value you want to send to that DAC value: a number between 0 and 255.

- Public Sub setAllDACh()

  This method will set all DA channels to the maximal value (255).

- Public Sub clearAllDACh()

  This method will clear all DA channels to the minimal value (0).

- Public Function getDAstate(channel as Integer)

  This function returns the value of a DA channel. The channel must be specified, and is an integer between 1 and 4.

3.1.4 AD Channels

- Public Function getADstate(channel as Integer)

  This function will return the value of an AD channel. The desired channel must be specified, and is a value between 1 and 16.

3.2 Control commands

Control commands are used to get some information from WinPLC.

- Public Sub hideWinPLC()

  Calling this method will hide WinPLC from your screen. You need to call the method showWinPLC() to get WinPLC back.

- Public Sub showWinPLC()

  Calling this method will show WinPLC on your screen. This only have affect if WinPLC was hidden.

- Public Sub closeWinPLC()

  Calling this method will close the opened WinPLC object. This method might trigger a runtime error. But you may ignore this error.

3.3 Runtime commands

Runtime commands can be used to manipulate a running WinPLC file. These commands can be used to interact with a program.

- Public Sub callRoutine(routinename As String)

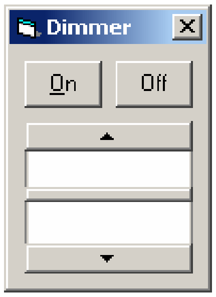
Calling the subroutine "CallRoutine" will only have affect when a WinPLC file is running in WinPLC. You can use this command to manipulate the execution of WinPLC: you can make the execution jump to a certain subroutine. After the subroutine is completed, WinPLC will proceed with its normal execution, and go back to the place where the execution was before this method call.

Like I mentionned before, this instruction has no effect if WinPLC is not running a program. The instruction also has no effect if you try to call a subroutine that is not present in the current program.

This is a very powerful function for WinPLC. I created a sample that demonstrates the use of this function. You can view this sample on page 18.

4 Programming examples

4.1 Visual Basic



The first WinPLC example is a dimmer program. The dimmer program is able to communicate with WinPLC and monitor/change the value of high precision analog output channel 1.

The program has 2 buttons: on and off. ON will set the DA channel to the maximum value, while OFF will set the DA channel to the minimal value. There is also a scrollbar that indicated the current value of the DA channel. Of course, the scrollbar value can also be changed.

Without the scrollbar, this application could be made perfectly with WinPLC and its GUI creator. The sample demonstrates the power of plugging your own graphical user interface into WinPLC. Besides the more powerfull GUI, there is a second advantage: you can run more than one of these programs at the same time.

Source Code

This is the source code for the Dimmer form in the Visual Basic application. The Application is written in Visual Basic 5.0 Professional.

```
' declare winplc object that can be accessed from every sub
Dim winplc As Object


Private Sub Form_Load()
    ' program is started, load winplc object

    ' load winplc object
    Set winplc = CreateObject("PWinPLC.PowerWinPLC")

    ' set scrollbar values
    VScroll1.Min = 255
    VScroll1.Max = 0
    VScroll1.Value = winplc.getDAState(1)
End Sub


Private Sub cmd_on_Click()
    ' set da channel 1 to maximum
    winplc.setdach 1, 255
    VScroll1.Value = winplc.getDAState(1)
End Sub


Private Sub cmd_off_Click()
    ' set da channel 1 to minimum
    winplc.setdach 1, 0
    VScroll1.Value = winplc.getDAState(1)
End Sub
```

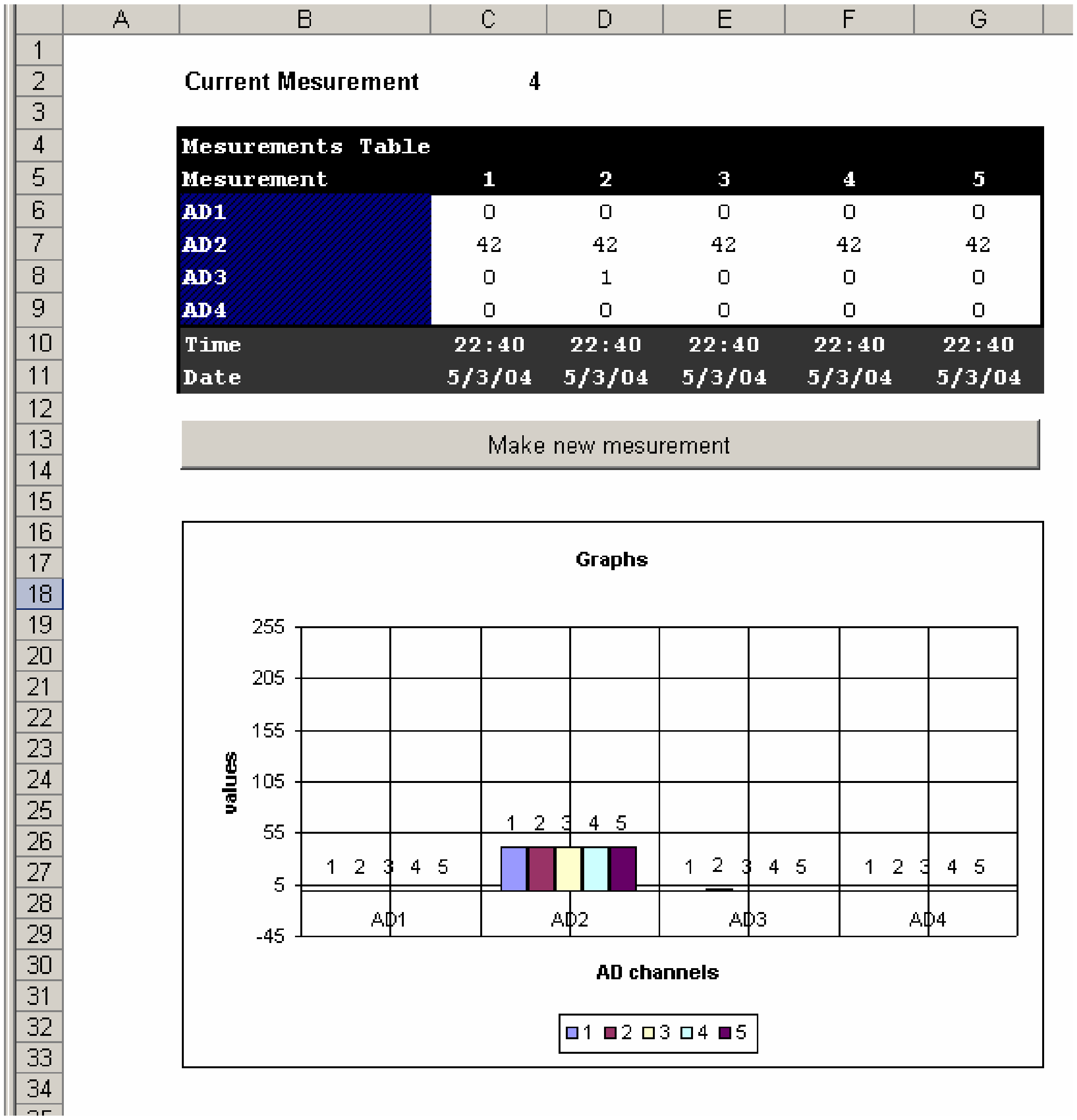
```
Private Sub VScroll1_Change()
        ' change dac channel when scrollbar is moving
        On Error Resume Next
        winplc.setdach 1, VScroll1.Value
    End Sub


Private Sub VScroll1_Scroll()
        ' change dac channel when scrollbar is moving
        On Error Resume Next
        winplc.setdach 1, VScroll1.Value
    End Sub

    Private Sub Timer1_Timer()
        ' update scrollbar value
        On Error Resume Next
        VScroll1.Value = winplc.getDAState(1)
    End Sub
```

4.2 Visual Basic For Applications / Excel

Since Excel is a good program to create graphs, I made an example to make graphs of the K8000 AD channels. I made a certain worksheet, with a table and a graph. There is also a button, that can be clicked.

Each click on the button will perform a mesurement of the 4 AD channels. The AD channels will be read, and the results will be placed in the table. Where? Well, the table allows to show 5 mesurements. Each time you perform a mesurement, the data will be written in the next column (in the order of 1 – 2 – 3 – 4 – 5). After column 5, column 1 will become overwritten.

For each mesurement, the current time and date will be displayed. The result will also be shown in the table called "Graphs".


Current Mesurement 4

Mesurements Table					
Mesurement	1	2	3	4	5
AD1	0	0	0	0	0
AD2	42	42	42	42	42
AD3	0	1	0	0	0
AD4	0	0	0	0	0
Time	22:40	22:40	22:40	22:40	22:40
Date	5/3/04	5/3/04	5/3/04	5/3/04	5/3/04

Source code

```
Private Sub cmd_make_Click()

    ' declaration section
    Dim current_mesurement As Integer
    Dim letter As String

    ' get current mesurement number
    Range("C2").Select
    current_mesurement = Val(ActiveCell.FormulaR1C1)

    ' deterimine the number
    Select Case current_mesurement
    Case "1": letter = "C"
    Case "2": letter = "D"
    Case "3": letter = "E"
    Case "4": letter = "F"
    Case "5": letter = "G"
    End Select

    ' make connection with K8000
    Dim K8000 As Object
    Set K8000 = CreateObject("PWinPLC.PowerWinPLC")

    ' get the value of the channels
    Dim i As Integer
    For i = 1 To 4
        Range(letter & (i + 5)).Select
        ActiveCell.FormulaR1C1 = K8000.getadstate(i)
    Next i

    ' show time & date info
    Range(letter & (i + 5)).Select
    ActiveCell.FormulaR1C1 = Time
    Range(letter & (i + 6)).Select
    ActiveCell.FormulaR1C1 = Date

    ' update mesurement counter
    current_mesurement = current_mesurement + 1
    If current_mesurement > 5 Then current_mesurement = 1
    Range("C2").Select
    ActiveCell.FormulaR1C1 = current_mesurement

End Sub
```

4.3 Visual Basic For Applications / Word

This sample for Microsoft Word illustrates a report. Once you open the document[2], it will automatically be generated by Word.

Source code[3] for report.doc

```
Sub autoOpen()
    Dim n As Integer
    Dim k8000 As Object
    Set k8000 = CreateObject("PWinPLC.PowerWinPLC")

    Selection.TypeText Text:="K8000 I/O report"
    Selection.HomeKey Unit:=wdLine, Extend:=wdExtend
    Selection.Font.Size = 18
    Selection.Font.Bold = wdToggle
    Selection.ParagraphFormat.Alignment = wdAlignParagraphCenter
    Selection.MoveDown Unit:=wdLine, Count:=1
    Selection.TypeParagraph
    Selection.TypeParagraph
    Selection.TypeParagraph
    Selection.ParagraphFormat.Alignment = wdAlignParagraphLeft
    Selection.TypeText Text:="Listing with the I/O channels:"
    Selection.TypeParagraph
    Selection.TypeParagraph

    For n = 1 To 16
        Selection.TypeText Text:="I/O " & n & ": " & Str(k8000.getIOState(n))
        Selection.TypeParagraph
    Next n
End Sub
```

This source code will generate a page with information on the I/O channels of the K8000 card. The code is executed automatically when you open the file. This is because the code is put in the autoOpen() sub. This is a subroutine that is called when Word is opening the document.

[2] This document uses Macro's. When you open the file, Word will give you a warning that the file might contain viruses. You can ignore the warning, and proceed with opening the file: it is NOT harmful.
[3] You can view the documents source code by pressing ALT+F11 in Word. This will open the Visual Basic editor for Word.

4.4 Windows VB Scripting

Window VB scripts are textfiles with a .vbs extension. They are used for purposes like maintanance. I have created 2 scripts: setallio.vbs and clearallio.vbs. setallio.vbs will set all I/O channels, while clearallio.vbs will clear all I/O channels.

Source code[4] for setallio.vbs

```
' create a new object
set winplc=createobject("PWinPLC.PowerWinPLC")
winplc.setAllIOch
set winplc=nothing
```

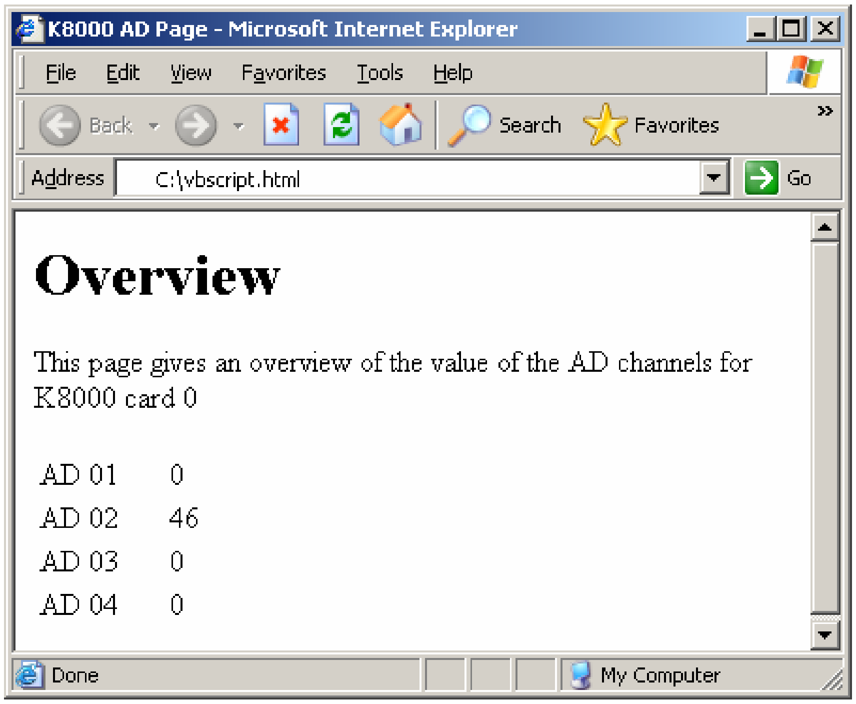
Source code for clearallio.vbs

```
' create a new object
set winplc=createobject("PWinPLC.PowerWinPLC")
winplc.clearAllIOch
set winplc=nothing
```

[4] You can just copy & paste the code into notepad, and save the code as a file with a .VBS extension. In Windows, you can just run the .VBS file by double clicking it in explorer.

4.5 HTML VB Script

The K8000 can be controlled from webpages that you run locally on your computer, if you are using Vbscript in those pages. Doing so, you can build a complete controlling system for your K8000 page. It's obviously you can't place those files on other websites, since they need to access components that are only available on **your** computer (client side programming). If you want other users to be able to control your WinPLC & K8000 from another computer, you will need to use something like ASP (Active Server Pages) and a webserver on your computer. This webserver can serve webpages that are available on your computer. These webpages can contain **server-side** ASP scripts, that access your computer: so users can make certain requests for your computer through a webpage, the code is executed on your computer, and the webserver sends the result to the user.



Overview

This page gives an overview of the value of the AD channels for K8000 card 0

AD 01	0
AD 02	46
AD 03	0
AD 04	0

Principe of the code

I first created a standard webpage, according to the XHTML strict specifications. The document contains a little text and a table. Then I added some VBScript coding: some code for the initialisation of the WinPLC Object (the communication between webpage and WinPLC).

Then I added some code that calls a function to return the values of the AD channels 1 to 4. This information is inserted into the HTML document. This information will be showed if you open the HTM file, and renewed if you refresh the HTM file.

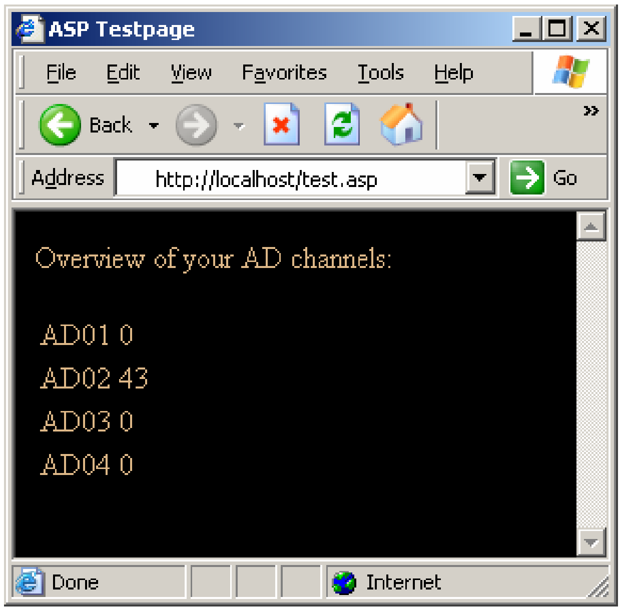
Source code for the webpage

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>K8000 AD Page</title>

<script type="text/vbscript">
       dim k8000
       set k8000=createobject("PWinPLC.PowerWinPLC")
</script>

</head>
<body>

<h1>Overview</h1>
<p>This page gives an overview of the value of the AD channels for K8000 card 0</p>
<table summary="i/o values" width="100">
<tr>
<td>AD 01</td>
<td><script type="text/vbscript">document.write k8000.getADState(1)</script></td>
</tr>
<tr>
<td>AD 02</td>
<td><script type="text/vbscript">document.write k8000.getADState(2)</script></td>
</tr>
<tr>
<td>AD 03</td>
<td><script type="text/vbscript">document.write k8000.getADState(3)</script></td>
</tr>
<tr>
<td>AD 04</td>
<td><script type="text/vbscript">document.write k8000.getADState(4)</script></td>
</tr>

<script type="text/vbscript">
       set k8000=nothing
</script>


</body>
</html>
```

4.6 HTML ASP Script

The previous HTML VBScript example showed that WinPLC can be controlled from webpages, using VBScript client-side programming. But there is an important restriction: the webpages can only be used on the computer where WinPLC is running.

ASP is the solution for this problem: it contains code that is executed on the server-side, not on the client-side. When the visitor of the page tries to view the pages, some code will be executed on the server (like interfacing with WinPLC, and exchange some data). The visitor will only see the result.


ASP Testpage
File Edit View Favorites Tools Help
Back
Address
http://localhost/test.asp
Go
Done
Internet

```
Overview of your AD channels:

AD01 0
AD02 43
AD03 0
AD04 0
```

Setting up the webserver

For realizing this test, I've used the IIS webserver from Microsoft (shipped with Microsoft Windows NT, Windows 2000, Windows XP Professional).

I experienced (and still am experiencing) some problems with the configuration of the server. This is because WinPLC was running under my Windows XP account, while the webserver is running under another account. So there were some communcation problems, and they are not all fixed yet.

Source code for test.asp

```
<%
' ASP Example file - By Vanderhaegen Bart
'
' bartv@pandora.be
'
' This page demonstrates how you can control the K8000 from a webpage,
' served on an IIS webserver (Microsoft Windows 2000, Windows XP). WinPLC
' must be installed correctly in order to get this sample to work. (some
' components need to be registered etc...).
'
' Place this file in one of the directories of your server!
%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<link rel="stylesheet" type="text/css" href="test.css" />
<title>ASP Testpage</title>
</head>
<body>

<p>Overview of your AD channels:</p>

<%
' initialize the component. (PowerWinPLC must be installed correctly, or this
' line will give you an error).
Set myObj = Server.CreateObject("PWinPLC.PowerWinPLC")
%>

<table>
<tr>
<td>AD01</td>
<td><% response.write(myObj.getADState(1)) %></td>
</tr>

<tr>
<td>AD02</td>
<td><% response.write(myObj.getADState(2)) %></td>
</tr>

<tr>
<td>AD03</td>
<td><% response.write(myObj.getADState(3)) %></td>
</tr>

<tr>
<td>AD04</td>
<td><% response.write(myObj.getADState(4)) %></td>
</tr>
</table>

<%
' close the object?
Set myObj=nothing
%>


</body>
</html>
```

4.7 Java

WinPLC can also be controlled from Java (under Windows). I made a little test class that demonstrates the use of Java: a little program will set all odd I/O channels when started.

Source code

```
import com.jacob.com.*;
import com.jacob.activeX.*;

public class Test {
    public static void main(String asArgs[]) throws Exception {
ActiveXComponent k8000Ax = new ActiveXComponent("PWinPLC.PowerWinPLC");
       Object k8000 = k8000Ax.getObject();

       for (int n=1;n<16;n+=2) {
              Dispatch.call(k8000, "setIOCh",new Variant(n));
       }
      }
}
```

The use of JACOB

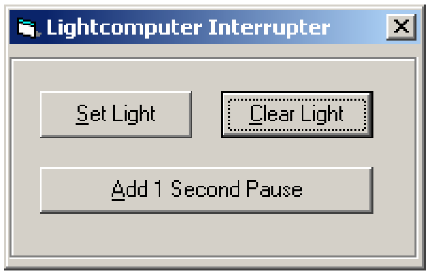
Normally, you can’t communicate between Java and ActiveX components. That’s where JACOB comes in: it allows you to make a bridge between Java and ActiveX. JACOB is available at http://danadler.com/jacob. You need to download these files to your computer, so you can use these extra classes to your Java project.

My test situation/Installing JACOB

I have created this example with RealJ and Jcreator (2 tools to create Java software). I am using the Java Development Kit from Sun, which is installed under the folder **C:\program files\jdk**.

I unpacked the file **jacobBin_17.zip** from the JACOB website, and placed the files **jacob.dll** and **jacob.rar** under the directory C:\Program Files\jdk\jre\lib\ext. Doing so, the needed libraries can always be found.

5 Sample on “CallRoutine”



This sample demonstrates the CallRoutine function. I have written an application in Visual Basic, which can be seen in the picture. The application contains 3 buttons: Set Light and Clear Light (they will set and clear the DA1 channel). There is also a button called “Add 1 Second Pause”.

This program assumes that WinPLC is running the **interrupt.plc** sample file. The code for this WinPLC program is shown below:

```
TIMESTEP = 150
TimeStep ON
LABEL WINPLC
     SETION 1000100010001000
     SETION 0100010001000100
     SETION 0010001000100010
     SETION 0001000100010001
GOTO WINPLC


SUB DO_INTERRUPT
     WAIT 1000
END SUB
```

The program has a endless loop (between the commands `LABEL WINPLC` and `GOTO WINPLC`). In this loop, the program sends some lighteffects to the K8000 card.

The program has also a subroutine called “`DO_INTERRUPT`”. As you can see, the subroutine will never be executed since there are no calls to it in the WinPLC program. But using the function “CallRoutine”, `DO_INTERRUPT` can be called from other programs.

In my Visual Basic sample program, `DO_INTERRUPT` will be called if you press the “Add 1 Second Pause” button. If you press the button once (while the WinPLC program runs), `DO_INTERRUPT` is called and will wait for 1000 ms (1 second). Afterwards, the subroutine will be exited, and the lightcomputer will resume with its effects.

If you press the “Add 1 Second Pause” button twice after each other, the total delay time will be 2 seconds: if you call a new routine before the current routine is finished, the new routine will run on top of the existing routine. This is nesting. You can nest up to 100 routines. That should be plenty.