PVI DLL Programming
TM711

Perfection in Automation
www.br-automation.com

Requirements

Training modules:	TM211 - Automation Studio Online Communication TM710 - PVI Communication
Software:	Visual Basic 6.0 Windows NT/2000/XP
Hardware:	any SG3 or SG4 controller

Table of contents

1. INTRODUCTION

This training module describes how to program the PVICOM.DLL using the PVI functions to create a Windows visualization or Service Tool.

Visual Basic 6.0 is used to describe the PVICOM.DLL functions and processes because the majority of applications are created in this programming language.

The training module "TM712 PVIServices" is recommended for applications created in Visual Studio.NET development environments.

Fig. 1: PVI DLL programming

1.1 Objective

Exercise examples will help participants to create PVI Client applications.

Participants will understand the application and possibilities of PVI functions.

The PVI user documentation will provide more in-depth knowledge of the PVI Client application possibilities.

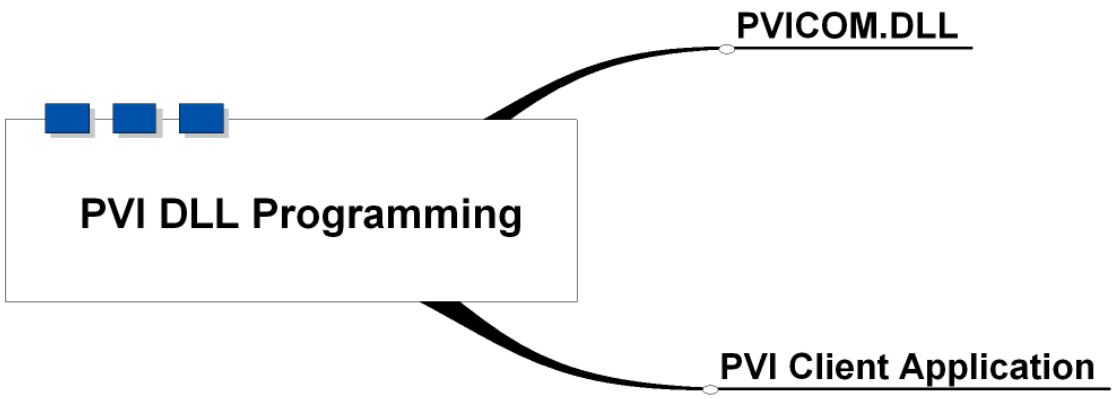


Fig. 2: Overview

2. PVICOM.DLL PROGRAMMING

The PVICOM interface is used by all Windows-based applications with PVI access.

This is the most optimum PVI interface in regard to performance.

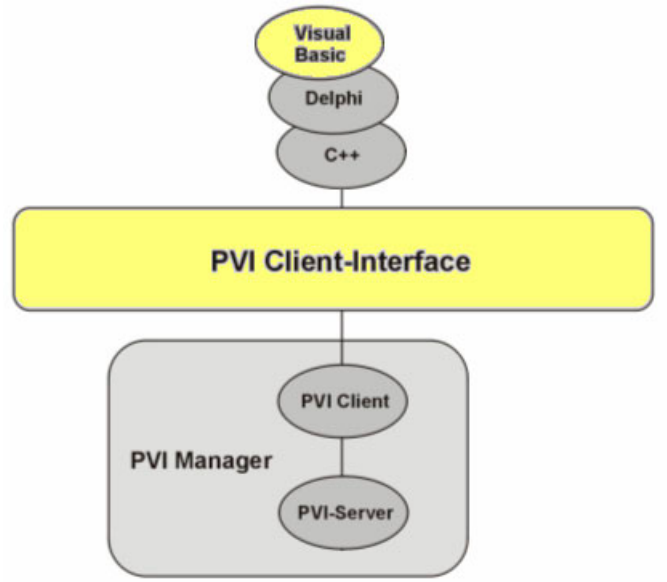


Fig. 3: PVI Client programming

Communication via the PVICOM interface is handled with the functions in the PVI communication library, "PviCom.dll".

The PVI communication library is a DLL (Dynamic Link Library) based on the Windows standard. If applications use functions of a DLL, then the DLL has to be loaded explicitly with functions being declared, or the static library "PviCom.lib" has to be bundled to the program (loaded implicitly).

The procedure can be found in the respective programming language manuals.

The following development environments are supported:

- Visual C++ (Version 6.0 and up)
- Visual Basic (Version 6.0 and up)
- Borland C++ Builder (Version 3 and up)
- Borland Delphi (Version 4 and up)

2.1 PVI installation files

The PVICOM definition files for the respective programming language are installed in the following directory after installing the PVI Server & Runtime / Development Package:

PVI Server&Runtime Setup:

BrAutomation\Pvi\Inc
BrAutomation\Pvi\Lib

PVI Development Setup:

Program Files\BrAutomation\Pvi\%Version%\Pvi\Inc
Program Files\BrAutomation\Pvi\%Version%\Pvi\Lib

The PVICOM definition file contains definitions and/or declaration for all PVICOM interface functions, types, structures used, and PVI constants.

File	Description
PviCom.lib	Static library for Visual C
PviCom.h	Definition file (include file) for Visual C
PviCom.bas	Definition file (module) for Visual Basic
PviCom.pas	Definition file (include file) for DELPHI

The communication library "**PviCom.dll**" is installed in the following directory when installing a PVI package:

Windows\System32

2.2 Requirements for PVI programming

AR000 is used for communication with a controller. The variables and the corresponding data type used to do this are documented in this training module.

2.2.1 The Automation Studio project

Automation Studio must be used to a user task in which a few variables are used. These variables are read and written in the PVI Client application that is created with this training module.

An existing Automation Studio project can also be used, but the variable names in this training module must be replaced accordingly.

The first step requires a task "**pvitest**" with the variables "**Lifesign**" and "**PV1**".

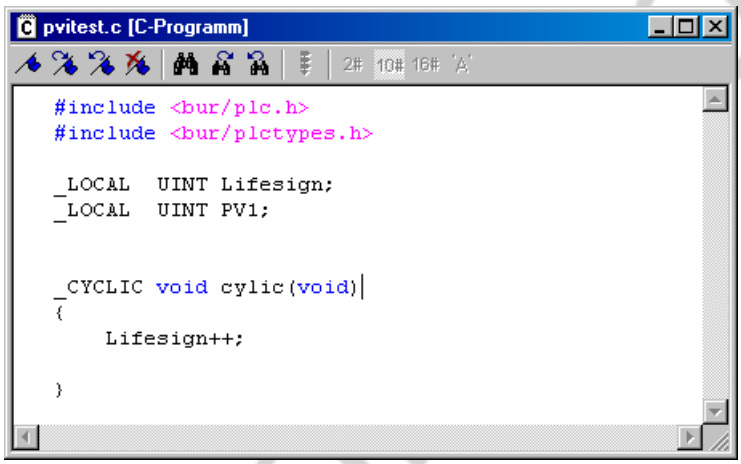


Fig. 4: Automation Studio test project

2.2.2 Visual Studio programming environment

All of the functions described in this training module are explained using Visual Basic 6.0 program code.

All PVI functions are described in the PVI user documentation using Visual C++ program code.

Additionally, a tutorial for Visual Basic, Visual C++, Borland Delphi and Borland C programmers is also placed in the "...\Pvi\Tutorial" directory during the PVI Server&Runtime / Development installation. The steps in the tutorial are described in the PVI user documentation.

3. PVI CLIENT APPLICATION

A small PVI Client application will be created step-by-step in this chapter. Exercises and tasks will be used to describe most of the PVI functions and their application.

What will be covered in this practice example:

- PVICOM functions
- Establish connection with the PVI Manager
- Setup the process objects with synchronous / asynchronous functions
- Evaluate the response data
- Read and write functions

3.1 PVICOM functions

For the first step, the definition file "**PVICOM.BAS**" will be added to the newly created project. Now it is possible to use PVI functions.

Exercise: Inserting the file PVICOM.BAS to the newly created VB project.

Select the module from the directory "...\Pvi\Inc\PviCom.bas".

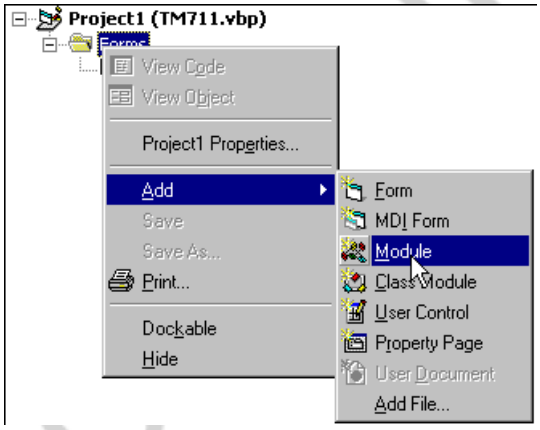


Fig. 5: Adding the PVICOM.BAS module

Select an existing module from the "Add Module" dialog box.

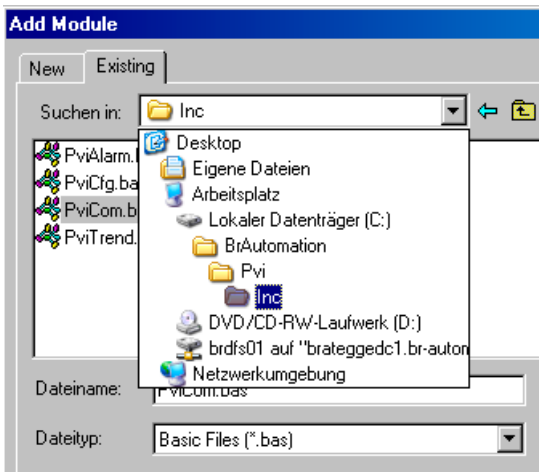


Fig. 6: Add Module – Add existing module.

Caution:

Only a reference to the file is created after inserting the module. The reference will not work anymore if the project directory is changed (e.g. passing the project on to someone else). In this case, it is recommended to copy the module to the VB project directory.

The PVICOM.BAS module is now displayed in the project explorer.

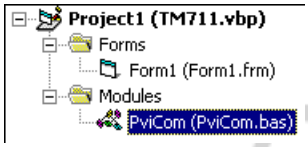


Fig. 7: Project explorer – PviCom.bas

All PVICOM constants, structures and function descriptions are contained in this module.

Caution:

The user is not allowed to make any changes to this module.

Descriptions of the functions can be found in the PVI user documentation.

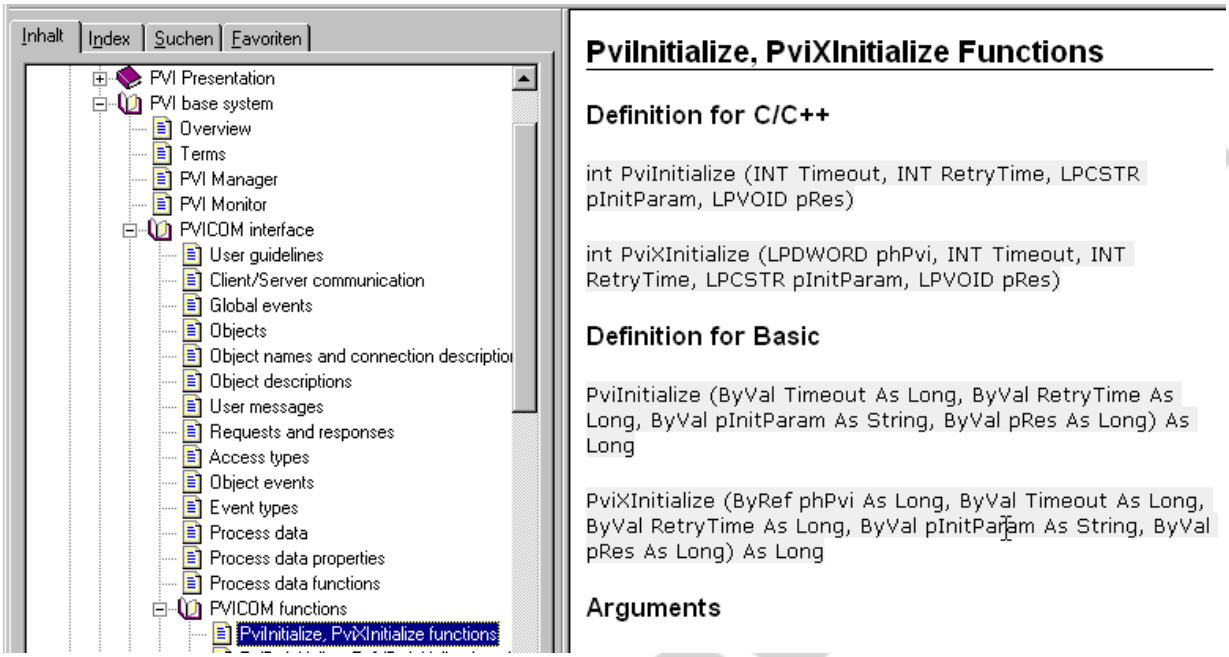


Fig. 8: PVICOM Help

As you can see already with the first function, there is always a Pvi... and PviX... function for the same task.

The Pvi... functions are sufficient for a simple PVI Client application. However, if the application will be accessing multiple server PCs on a distributed network, then the PviX... functions must be used.

This training module only uses the Pvi... functions.

3.2 Establish a connection with the PVI Manager

The connection with the PVI Manager is established using the "**PviInitialize**" function.

Exercise: Establish and then cancel a connection with the PviManager.

The "**PviInitialize()**" function and its parameters are used in the form's "**Load**" event. The "**PviDeinitialize()**" function is called in the "**Unload**" event.

The function's arguments can be found in the PVI user documentation.

```
Private Sub Form_Load()
Dim TimeOut As Long
Dim RetryTime As Long
Dim InitParameter As String

    TimeOut = 0      ' Default timeout 30 s
    RetryTime = 0    ' no retry (default)
    InitParameter = "LM=0"
    PviInitialize TimeOut, RetryTime, InitParameter
End Sub
```

```
Private Sub Form_Unload(Cancel As Integer)
    PviDeinitialize
End Sub
```

This example shows local communication between the PVI application and the PVI Manager. A remote connection is transferred using the parameters **PN** and **IP** with the argument "**InitParameter**".

InitParameter = "LM=0 PT=0 PN=20000 IP=HostName"

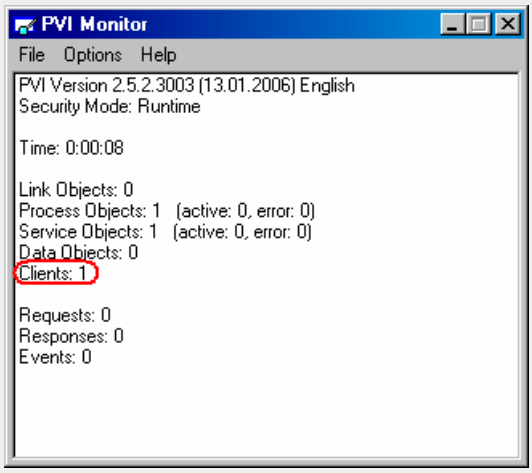
Caution:

The "PviDeinitialize" function should always be used. Communication between the application and the PVI Manager is cancelled at this point. The application could crash if this function is not called because accumulated callbacks can no longer be processed.

Result:

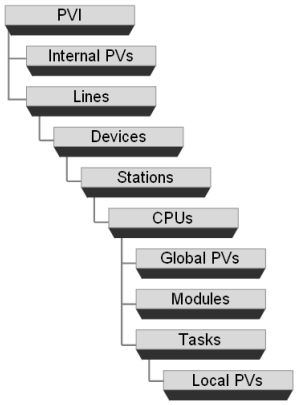
The **PVI Manager is started** after starting the application. PVI objects can then be created or accessed on already existing PVI objects.

If the PVI Manager is already running, then the application is registered on the PVI Manager – the PVI Manager detects a new Client. Existing PVI objects (static) can be accessed.



3.3 Setting up the process objects

The process objects can be put together according to the PVI object hierarchy once the application has established a connection with the PVI Manager.



When doing this, you should determine whether these objects will be set up as static or temporary before creating the objects (see TM710 or the PVI user documentation).

Functions used:

- PviCreate / PviCreateRequest
- PviLink / PviLinkRequest

3.3.1 Setting up temporary process objects

The process object is created including the link object (Create + Link) when setting up a temporary process object. When disconnecting the link object (unlink) or when terminating the PVICOM application, the temporary process object is also disconnecting again (i.e. deleted on the PVI Manager).

Exercise: Creating temporary process objects with synchronous PVI functions.

All process objects up to and including variable objects are set up and the response data (see also 3.4) from the "**Lifesign**" variable is displayed in a textbox.

Steps for creating the process objects:

- Add a module with the name "**myPviFunc**" for creating the global variables and the callback functions for evaluating the data
- Place a button on the form with the name "**cmdCreateTempSync**"
- Place a textbox with the name "**txtLifesign**"
- Create the **callback function** in the module "**myPviFunc**"

A **variable** must be set up in the "**myPviFunc**" module for each PVI object. These variables are required for **returning the handle** (reference) of a PVI object. This handle can be used for read or write access to a PVI object during runtime.

```
Option Explicit

Public hLine As Long
Public hDevice As Long
Public hStation As Long
Public hCpu As Long
Public hTask As Long
Public hPV_Lifesign As Long
```

The following code is entered to the button's "**Click**" event:

```
Private Sub cmdCreateSync_Click()
Dim ReturnVal As Long        ' Returnvalue of each function
Dim ObjName As String        ' variable for process object name
Dim ObjDescription As String ' variable for connection descr.
Dim LinkDescription As String ' variable for link object

    ' Create Line Object
    ObjName = "@/Pvi/LnIna2"
    ObjDescription = "CD=" & """" & "LnIna2" & """"
    LinkDescription = vbNullString

    ReturnVal = PviCreate(hLine, ByVal ObjName, POBJ_LINE, ByVal _
                ObjDescription, AddressOf PviCallback, _
                SET_PVICALLBACK, 0, LinkDescription)
    If ReturnVal Then
        MsgBox "Error create Line object E#" & ReturnVal
        Exit Sub
    End If


End Sub
```

Description of the setup procedure:

- Variables are created for the object name (path name), the connection description and the link object.
- The variables are set with the path name and the connection description of the respective process object. The link object does not have to be preset for the line object.
- Call the synchronous function "**PviCreate**".
- The global variable "hLine" is transferred as handle with the object type "POBJ_LINE".
- A temporary process object is set up by specifying the **callback** function with "**AddressOf PviCallback**". A static object would be set up if "NULL" were transferred for these arguments.
- By specifying the callback without the data "**SET_PVICALLBACK**", the data in the callback function must be read with "**PviReadResponse**".
- "0" is transferred as object number because response data is not necessary on this object. This is first evaluated and transferred starting at the CPU object.

Note:

The program cannot be started until the callback function has also been set up. This item "evaluating the response data" is now anticipated, a function is created where the response data from each PVI object is evaluated.

The callback function is used without data in the "**myPviFunc**" module.

```
Public Sub PviCallback(ByVal WPARAM As Long, ByVal LPARAM As Long)
Dim ReturnVal As Long

    Select Case LPARAM


        Case Else
            PviReadResponse WPARAM, 0, 0


    End Select
End Sub
```

The message sent by the PVI Manager is acknowledge by calling the "**PviReadResponse**" function.

Caution:

All messages from the PVI Manager must be acknowledged in the callback, even if no PVI object data or errors are evaluated.

Task: Start the program and set up the line object.

After starting, the **result should be tested in the PVI Monitor** or in the PVI SnapShot Viewer.

Result:

Two process objects are displayed in the PVI Monitor (PVI object + line object). A link object is also set up at the same time as the line object.

```
Link Objects: 1
Process Objects: 2  (active: 1, error: 0)
```

If the application is closed now, the process object including link object is also deleted.

The following program code is added to the function in the "**cmdCreateSync**" button - click event:

```
' Create Device Object
ObjName = "@/Pvi/LnIna2/TCPIP"
ObjDescription = "CD=" & """ & "/IF=TCPIP /SA=1" & """
LinkDescription = vbNullString

ReturnVal = PviCreate(hDevice, ByVal ObjName, POBJ_DEVICE, ByVal _
            ObjDescription, AddressOf PviCallback, _
            SET_PVICALLBACK, 0, LinkDescription)
If ReturnVal Then
    MsgBox "Error create Device object E#" & ReturnVal
    Exit Sub
End If

' Create Station Object
ObjName = "@/Pvi/LnIna2/TCPIP/Station"
ObjDescription = "CD="
LinkDescription = vbNullString

ReturnVal = PviCreate(hStation, ByVal ObjName, POBJ_STATION, ByVal _
            ObjDescription, AddressOf PviCallback, _
            SET_PVICALLBACK, 0, LinkDescription)
If ReturnVal Then
    MsgBox "Error create Station object E#" & ReturnVal
    Exit Sub
End If
```

As you can see here, the path name (ObjName) is expanded for each object. The variable for the handle and the object type also change for each process object.

Further in the program code:

```
' Create CPU Object
ObjName = "@/Pvi/LnIna2/TCPIP/Station/CPU"
ObjDescription = "CD=" & """" & "/DA=2 /DAIP=127.0.0.1 /REPO=11160" & """"
LinkDescription = "EV=ed"

ReturnVal = PviCreate(hCpu, ByVal ObjName, POBJ_CPU, ByVal _
            ObjDescription, AddressOf PviCallback, _
            SET_PVICALLBACK, 1, LinkDescription)
If ReturnVal Then
    MsgBox "Error create Device object E#" & ReturnVal
    Exit Sub
End If
```

Response data should be evaluated when creating the CPU object. To do this, the event mask "**EV**" is set up for "**e=Error**" and "**d=Data**" in the link description (variable LinkDescription). That means that the error is registered in the callback for the user or object number "**1**" if the connection to the controller is lost.

Task: Expanding the callback and evaluating a link error:

The user number is transferred to the callback in the "**LPARAM**" argument.

```
Public Sub PviCallback(ByVal WPARAM As Long, ByVal LPARAM As Long)
Dim ReturnVal As Long

    Select Case LPARAM

        Case 1:      ' CPU object

            ReturnVal = PviReadResponse(WPARAM, 0, 0)
            If ReturnVal Then    ' error detected
                Form1.Caption = "E#" & ReturnVal
            Else
                Form1.Caption = "Connection to PLC OK"
            End If

        Case Else
            PviReadResponse WPARAM, 0, 0


    End Select
End Sub
```

A CPU object error is evaluated in the callback function with **LPARAM = 1** and then written to the form's caption. The caption also shows when the connection to the controller has been established.

Task:Add the task and variable object and evaluate the data changes and errors in the callback function.

The task object for the "**pvitest**" user task and the "**Lifesign**" variable are set up.

```
' Create Task Object
ObjName = "@/Pvi/LnIna2/TCPIP/Station/CPU/Task1"
ObjDescription = "CD=" & """ & "pvitest" & """
LinkDescription = vbNullString

ReturnVal = PviCreate(hTask, ByVal ObjName, POBJ_TASK, ByVal _
            ObjDescription, AddressOf PviCallback, _
            SET_PVICALLBACK, 0, LinkDescription)
If ReturnVal Then
    MsgBox "Error create Task object E#" & ReturnVal
    Exit Sub
End If

' Create Variable Object
ObjName = "@/Pvi/LnIna2/TCPIP/Station/CPU/Task1/PV_Life"
ObjDescription = "CD=" & """ & "Lifesign" & """ & " AT=rw RF=250"
LinkDescription = "EV=ed VT=f64"

ReturnVal = PviCreate(hPV_Lifesign, ByVal ObjName, POBJ_PVAR, ByVal _
            ObjDescription, AddressOf PviCallback, _
            SET_PVICALLBACK, 10, LinkDescription)
If ReturnVal Then
    MsgBox "Error create Variable object E#" & ReturnVal
    Exit Sub
End If
```

The attribute for read and write access "**AT=rw**" as well as the refresh time "**RF=250**" (in [ms]) are also specified in the connection description (CD=) of the variable object.

The event mask "**EV**" is set up for "**e=Error**" and "**d=Data**" in the description of the link object. A cast to a "**Double**" data type is performed using the parameter "**VT=f64**".

When changing the type, a controller data type is converted to a data type used in the VB.

This makes it possible to also display and process a UDINT data type from a control variable in Visual Basic in its entire value range.

Data type	Value range
Automation Runtime – UDINT	0 - 4294967295
Visual Basic – LONG	-247483648 - 247483647

Caution:

The value range of the data type used in VB is not automatically monitored when writing. Therefore, the data function = scaling had to be used with the parameter "**FS**".

The variable's data is written to the text property of the "**txtLifesign**" textbox in the callback. The error number is output to the text box when return value <> 0 comes from the response function.

```
Public Sub PviCallback(ByVal WPARAM As Long, ByVal LPARAM As Long)
Dim ReturnVal As Long
Dim DataPV As Double

    Select Case LPARAM

        Case 1:     ' CPU object

            ReturnVal = PviReadResponse(WPARAM, 0, 0)
            If ReturnVal Then    ' error detected
                Form1.Caption = "E#" & ReturnVal
            Else
                Form1.Caption = "Connection to PLC OK"
            End If
        Case 10:     ' variable Lifesign
            ReturnVal = PviReadResponse(WPARAM, DataPV, 8)
            If ReturnVal Then
                Form1.txtLifesign.Text = "E#" & ReturnVal
            Else
                Form1.txtLifesign.Text = DataPV
            End If
        Case Else
            PviReadResponse WPARAM, 0, 0
    End Select
End Sub
```

When the project is started and the button for creating the process object is pressed, the connection to the controller is established and the value of the variable is output to the textbox.

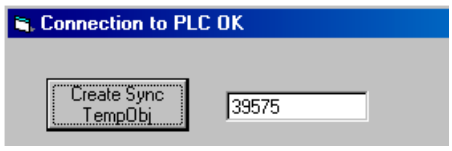


Fig. 9: Starting the application – Displaying the value

Task: Test a loss of connection with the controller

Cancel the connection to the controller. If it is being used, the AR000 must first be exited and the effects must be tested on a running VB program. With other types of communication, the connection cable must be disconnected from the controller.

The connection is automatically re-established once the connection is made again or the AR000 has been restarted.

Result:

The variable object is automatically read by the PVI Manager (active process object) when using the event mask "**EV=ed**".

The callback is automatically called only for active process objects each time a value change occurs and in the event of an error.

As a result, it is not necessary to read cyclically from the application.

Error evaluation does not have to be specially programmed. Instead, the application is automatically notified by the connection with the controller and via errors in the project setup.

The process objects on the PVI Manager are also deleted when ending the PVI Client application.

3.3.2 Setting up static process objects

A static process object is only set up (created) once and remains active throughout the entire runtime of PVI Manager. As many link objects as necessary can be linked to this static process object and unlinked again.

Exercise: Creating static process objects with asynchronous PVI functions.

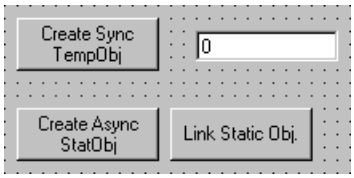
All process objects are set up into a variable object and the response data from the "**Lifesign**" variable is displayed in a textbox.

When setting up a static process object, a PviCreate (in this case the asynchronous call "**PviCreateRequest()**") is used without specifying the callback.

This is specified with "**PviLink()**" when the link object is set up.

Steps for creating the process objects:

- The same global variables are used for the object references (handles) as for the temporary process objects
- Pace a button on the form with the name "**cmdCreateStatAsync**"
- Pace a button on the form with the name "**cmdLinkStatObj**"
- Create a separate callback for the event data and for response data



Static process objects are not set up with asynchronous PVI functions in the "**cmdCreateStatAsync**" button's "**Click**" event.

```
Private Sub cmdCreateStatAsync_Click()
Dim ReturnVal As Long        ' Returnvalue of each function
Dim ObjName As String        ' variable for process object name
Dim ObjDescription As String ' variable for connection descr.
Dim LinkDescription As String ' variable for link object

    ' Create Line Object
    ObjName = "@/Pvi/LnIna2"
    ObjDescription = "CD=" & """" & "LnIna2" & """"

    ReturnVal = PviCreateRequest(ByVal ObjName, POBJ_LINE, ByVal _
                ObjDescription, 0, 0, 0, 0, AddressOf RespCreate, _
                SET_PVICALLBACK, 1)
    If ReturnVal Then
        MsgBox "Error create Line object E#" & ReturnVal
        Exit Sub
    End If

End Sub
```

The PviCreateRequest function requires two callback functions. One for the response from **Create** and one for evaluating the **response data and error** from the process object.

If the callback is only specified for "Create", then a static process object is created.

A callback is not required for the synchronous function PviCreate because the response to "Create" is contained in the function call.

In the next step we will create the callback function in the "**myPviFunc**" module for evaluating the Create function.

```
Public Sub RespCreate(ByVal WPARAM As Long, ByVal LPARAM As Long)
Dim ReturnVal As Long

    Select Case LPARAM
        Case 1:          ' Line object
            ReturnVal = PviCreateResponse(WPARAM, hLine)

        Case Else
            PviCreateResponse WPARAM, 0

    End Select
End Sub
```

A **PviCreateRequest** is acknowledged with **PviCreateResponse** in the callback for the response from the function call.

Note:

Error evaluation is not covered in this example. The user can implement this individually.

Task: Start the program and check the process objects in the PVI Monitor

Two process objects, the PVI and the Line object are set up. Unlike the temporary process objects, there is still no link object.

```
Link Objects: 0
Process Objects: 2  (active: 0, error: 0)
Service Objects: 2  (active: 0, error: 0)
Data Objects: 0
Clients: 1
```

Further in the program code for creating all process objects:

```
' Create Device Object
ObjName = "@/Pvi/LnIna2/TCPIP"                    _
ObjDescription = "CD=" & """ & "/IF=TCPIP /SA=1" & """

ReturnVal = PviCreateRequest(ByVal ObjName,POBJ DEVICE, ByVal _
            ObjDescription, 0, 0, 0, 0, AddressOf RespCreate, _
            SET_PVICALLBACK, 2)
If ReturnVal Then
    MsgBox "Error create Device object E#" & ReturnVal
    Exit Sub
End If
```

```
' Create Station Object
ObjName = "@/Pvi/LnIna2/TCPIP/Station"
ObjDescription = "CD="

ReturnVal = PviCreateRequest(ByVal ObjName, POBJ_STATION, ByVal _
            ObjDescription, 0, 0, 0, 0, AddressOf RespCreate, _
            SET_PVICALLBACK, 3)
If ReturnVal Then
    MsgBox "Error create Station object E#" & ReturnVal
    Exit Sub
End If

' Create CPU Object
ObjName = "@/Pvi/LnIna2/TCPIP/Station/CPU"
ObjDescription = "CD=" & """ & "/DA=2 /DAIP=127.0.0.1 /REPO=11160" & """

ReturnVal = PviCreateRequest(ByVal ObjName, POBJ_CPU, ByVal _
        ObjDescription, 0, 0, 0, 0, AddressOf RespCreate, _
        SET_PVICALLBACK, 4)

If ReturnVal Then
    MsgBox "Error create CPU object E#" & ReturnVal
    Exit Sub
End If

' Create Task Object
ObjName = "@/Pvi/LnIna2/TCPIP/Station/CPU/Task1"
ObjDescription = "CD=" & """ & "pvitest" & """

ReturnVal = PviCreateRequest(ByVal ObjName, POBJ_TASK, ByVal _
        ObjDescription, 0, 0, 0, 0, AddressOf RespCreate, _
        SET_PVICALLBACK, 5)

If ReturnVal Then
    MsgBox "Error create Task object E#" & ReturnVal
    Exit Sub
End If

' Create Variable Object
ObjName = "@/Pvi/LnIna2/TCPIP/Station/CPU/Task1/PV_Life"
ObjDescription = "CD=" & """ & "Lifesign" & """ & " AT=rw RF=250"

ReturnVal = PviCreateRequest(ByVal ObjName, POBJ_PVAR, ByVal _
    ObjDescription, 0, 0, 0, 0, AddressOf RespCreate, _
    SET_PVICALLBACK, 10)

If ReturnVal Then
    MsgBox "Error create Task object E#" & ReturnVal
    Exit Sub
End If
```

The callback for the CreateResponse function is also expanded:

```
Public Sub RespCreate(ByVal WPARAM As Long, ByVal LPARAM As Long)
Dim ReturnVal As Long

    Select Case LPARAM
        Case 1:          ' Line object
            ReturnVal = PviCreateResponse(WPARAM, hLine)
        Case 2:          ' Device object
            ReturnVal = PviCreateResponse(WPARAM, hDevice)
        Case 3:          ' Station object
            ReturnVal = PviCreateResponse(WPARAM, hStation)
        Case 4:          ' CPU object
            ReturnVal = PviCreateResponse(WPARAM, hCpu)
        Case 5:          ' Task object
            ReturnVal = PviCreateResponse(WPARAM, hTask)
        Case 10:         ' Variable object
            ReturnVal = PviCreateResponse(WPARAM, hPV_Lifesign)
        Case Else
            PviCreateResponse WPARAM, 0

    End Select
```

Task: Start the program and check the process objects in the PVI Monitor

All process objects are displayed in the PVI Monitor after starting the program.

```
Link Objects: 0
Process Objects: 7  (active: 0, error: 0)
Service Objects: 6  (active: 0, error: 0)
Data Objects: 0
Clients: 1
```

Result:

The process objects remain on the PVI Manager even when the application is ended.

A link object is not created for any process objects.

If the program is started again, then the **error 12002** is returned at the **PviCreateResponse**.

That means that the **object name already exists**. In this case, a link object can be set up immediately on the existing process object.

Caution:

If a static or temporary process object is set up by two different applications with the same path names (@/Pvi/Lnlna2...) but with different connection descriptions, then the connection description of the existing process object is used.

That means that it is not possible if a second application has the same path name and must communicate with e.g. another device.

When setting up an application with static objects you should also be aware that a change to the connection description due to a program or configuration error does not take effect until after terminating the PVI Manager.

3.3.3 Creating a link object

A link object with the function "**PviLink()**" or "**PviLinkRequest()**" must be created for a static process object. As many link objects as necessary can be connected to the same process object.

> **Note:**
>
> It is recommended to use the asynchronous "**PviLinkRequest()**" functions because they can also be called in a loop. This also significantly speeds up the processing of the functions for simultaneous linking and unlinking multiple link objects.

Each of these link objects can have different parameters e.g. for the data type or scaling. This makes it possible for example to simultaneously process a control variable in the application as raw value (i.e. the physical value of the controller) as well as the scaled and converted process value.

Exercise: Create the link objects up to the task object

The link objects are set up in the **click** even of the "**cmdLinkStatObj**" button.

The same callback function is used as the one for the temporary objects "**PviCallback**".

```
Private Sub cmdLinkStatObj_Click()
Dim ReturnVal As Long        ' Returnvalue of each function
Dim ObjName As String        ' variable for process object name
Dim LinkDescription As String ' variable for link object

    ' Link Line Object
    ObjName = "@/Pvi/LnIna2"
    LinkDescription = vbNullString

    ReturnVal = PviLink(hLine, ByVal ObjName, AddressOf PviCallback, _
                SET_PVICALLBACK, 0, LinkDescription)
    If ReturnVal Then
        MsgBox "Error link Line object E#" & ReturnVal
        Exit Sub
    End If
```

```
    ' Link Device Object
    ObjName = "@/Pvi/LnIna2/TCPIP"
    LinkDescription = vbNullString

    ReturnVal = PviLink(hDevice, ByVal ObjName, AddressOf PviCallback, _
                SET_PVICALLBACK, 0, LinkDescription)
    If ReturnVal Then
        MsgBox "Error link Device object E#" & ReturnVal
        Exit Sub
    End If

    ' Link Station Object
    ObjName = "@/Pvi/LnIna2/TCPIP/Station"
    LinkDescription = vbNullString

    ReturnVal = PviLink(hStation, ByVal ObjName, AddressOf PviCallback, _
                SET_PVICALLBACK, 0, LinkDescription)
    If ReturnVal Then
        MsgBox "Error link Station object E#" & ReturnVal
        Exit Sub
    End If

    ' Link CPU Object
    ObjName = "@/Pvi/LnIna2/TCPIP/Station/CPU"
    LinkDescription = "EV=ed"

    ReturnVal = PviLink(hCpu, ByVal ObjName, AddressOf PviCallback, _
                SET_PVICALLBACK, 1, LinkDescription)
    If ReturnVal Then
        MsgBox "Error link Device object E#" & ReturnVal
        Exit Sub
    End If

    ' Link Task Object
    ObjName = "@/Pvi/LnIna2/TCPIP/Station/CPU/Task1"
    LinkDescription = vbNullString

    ReturnVal = PviLink(hTask, ByVal ObjName, AddressOf PviCallback, _
                SET_PVICALLBACK, 0, LinkDescription)
    If ReturnVal Then
        MsgBox "Error link Task object E#" & ReturnVal
        Exit Sub
    End If

End Sub
```

As with the temporary process objects, the event mask is activated in the LinkDescriptor for errors and data "**EV=ed**" in the CPU object.

If the program is started now and the buttons "**cmdCreateStatAsync**" and "**cmdLinkStatObj**" are pushed consecutively, then a connection to the task object is established. All process objects (PVI- up to variable object) and link objects (PVI- up to task object) are displayed in the PVI Monitor.

```
Link Objects: 5
Process Objects: 7  (active: 5, error: 0)
Service Objects: 6  (active: 4, error: 0)
Data Objects: 0
Clients: 1
```

Exercise: Create the link object to the variable's process object

The process object is activated by linking to the variable object (event mask "**EV=ed**" in the LinkDescriptor). From this point on, the variable is read by the Manager and displayed in the "**txtLifesign**" textbox.

A new "**cmdLinkVar**" button should be created and a PviLink() to the variable object should be established in the click event.

```
Private Sub cmdLinkVar_Click()
Dim ReturnVal As Long        ' Returnvalue of each function
Dim ObjName As String        ' variable for process object name
Dim LinkDescription As String ' variable for link object

    ' Link Variable Object
    ObjName = "@/Pvi/LnIna2/TCPIP/Station/CPU/Task1/PV_Life"
    LinkDescription = "EV=ed VT=f64"

    ReturnVal = PviLink(hPV_Lifesign, ByVal ObjName, AddressOf PviCallback, _
                SET_PVICALLBACK, 10, LinkDescription)

    If ReturnVal Then
        MsgBox "Error link Variable object E#" & ReturnVal
        Exit Sub
    End If

End Sub
```

After restarting the program, the variable's link object can be created by pressing the button "**cmdLinkVar**" without creating the static process objects because the process objects up to the task object already exist.

The variable is read by the PVI Manager and displayed on the form's textbox.

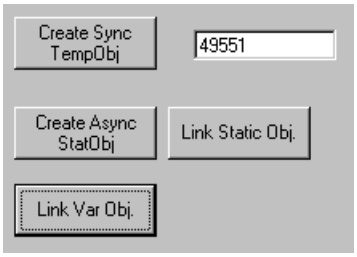


Fig. 10: Link to variable object

Exercise: Create multiple link objects to the same variable object

If the "**cmdLinkVar**" button is now pressed multiple times, then multiple link objects are created on the variable's process object.

This can be seen in the PVI Monitor with the number of **link objects**.

```
Link Objects: 5
Process Objects: 7   (active: 1, error: 0)
Service Objects: 6   (active: 1, error: 0)
Data Objects: 0
Clients: 1
```

Result:

Static process objects are set up one time and stay the same while the PVI Manager is running, even if the application is ended.

The process object is activated when one ore more link objects have been created.

3.3.4 Deleting a link object

The link to the process object is unlinked by "**deleting**" the link object with "**PviUnlink()**" or "**PviUnlinkRequest()**".

If there are multiple link objects on the same process object, then it is not "**inactive**" until the last link object has been deleted.

Exercise: Delete the link object for the variable object

The link object is deleted using the function "**PviUnlink()**".

A new button with the name "**cmdUnlinkVar**" should be created and the function should be called in the click event. The handle (i.e. the reference of the variable object) should be transferred to the function "**hPV_Lifesign**".

```
Private Sub cmdUnlinkVar_Click()
Dim returnval As Long

    returnval = PviUnlink(hPV_Lifesign)
    If returnval Then
        txtLifesign.Text = "E#" & returnval
    End If
End Sub
```

After starting the program, one or more link objects can be created by pressing the "**cmdLinkVar**" button. The value of the "**Lifesign**" variable is displayed in the textbox and refreshed cyclically.

The link object is deleted by pressing the "**cmdUnlinkVar**" button. The variable is no longer read once the last link object has been deleted.

An error is output by **evaluating the return value** from the "**PviLink()**" if the function is called when there are no more link objects.

3.4 Evaluating the response data

Two different types of callback functions can be used in Visual Basic.

- Callback without data
- Callback with data

The user messages are signaled with "**Post Messages**" for programming environments that support Window Messages.

"**Asynchronous callbacks**" are another possibility for evaluating response data. This method is used for Visual C++ applications without windows (DLL driver).

3.4.1 General information about evaluating response data

In general, you should make sure that the response data or event data is not evaluated in the respective callback / Window Message (e.g. access to databases).

> **Caution:**
>
> Within a callback function, unrestricted asynchronous PVICOM functions can be used, but synchronous PVICOM functions **CANNOT**.

3.4.2 Callback without data

In the previous exercises, the callback function was used without data.

With this type of callback, the data must be read with the corresponding Pvi...Response() function – the "**PviCreateResponse()**" function must be called in the callback for the "**PviCreateRequest()**" or "**PviCreate()**" function.

The "**WPARAM**" and "**LPARAM**" arguments transferred in the function are required to arrange the response function assigned to the request function.

```
Public Sub PviCallback(ByVal WPARAM As Long, ByVal LPARAM As Long)
Dim returnval As Long
Dim DataPV As Double

    Select Case LPARAM

        Case 1:     ' CPU object
            returnval = PviReadResponse(WPARAM, 0, 0)
        Case 10:    ' variable object
            returnval = PviReadResponse(WPARAM, DataPV, 8)
        Case Else:
            PviReadResponse WPARAM, 0, 0

    End Select

End Sub
```

The "**LPARAM**" argument indicates the user number specified in the respectively called function (e.g. PviCreate).

```
returnval = PviLink(hPV_Lifesign, ByVal ObjName, AddressOf PviCallback, _
                    SET_PVICALLBACK, 10, LinkDescription)
```

Caution:

If the return value that was transferred in the response function is <> 0, then an error has occurred and the data contained in the function is not valid.
Therefore, it is recommended to always evaluate the return value!

3.4.3 Callback with data

In this type of callback function, the response data and event data are transferred together in the callback function. Calling the corresponding response function is no longer necessary or possible.

The CallbackData is transferred with the function call instead of the normal callback function.

```
ReturnVal = PviLink(hPV_Lifesign, ByVal ObjName, AddressOf
                    PviCallbackData, SET_PVICALLBACK_DATA, 10, LinkDescription)


Public Sub PviCallbackData(ByVal WPARAM As Long, ByVal LPARAM As Long,
        ByVal pData As Long, ByVal dataLen As Long, _
        ByRef pResInfo As T_RESPONSE_INFO)

End Sub
```

Exercise: Use the callback with data to evaluate the event data for the "Lifesign" variable

The callback with data is used in the "**cmdLinkVar**" button's click event instead of the normal callback.

The callback function "**PviCallbackData**" is created in the "**myPviFunc**" module.

```
'ReturnVal = PviLink(hPV_Lifesign, ByVal ObjName, AddressOf PviCallback, _
             SET_PVICALLBACK, 10, LinkDescription)
ReturnVal = PviLink(hPV_Lifesign, ByVal ObjName, AddressOf PviCallbackData, _
             SET_PVICALLBACK_DATA, 10, LinkDescription)
```

In the callback function with data, a query is made in the "**T_RESPONSE_INFO**" **PVI structure** asking whether this is data or if an error has occurred. The data is then valid and can be evaluated.

```
Public Sub PviCallbackData(ByVal WPARAM As Long, ByVal LPARAM As Long, _
            ByVal pData As Long, ByVal dataLen As Long, _
            ByRef pResInfo As T_RESPONSE_INFO)

    Select Case LPARAM

        Case 10:
            If pResInfo.nMode = POBJ_MODE_EVENT Then
                If pResInfo.ErrCode Then
                    Form1.txtLifesign.Text = "E#" & pResInfo.ErrCode
                Else
                    GetDataInformation pData, dataLen
                End If
            End If
    End Select
End Sub
```

The corresponding process object is assigned again when evaluating the user parameter "**LPARAM**".

The "**nMode**" Member in the "**T_RESPONSE_INFO**" structure evaluates whether or not it is a data event (**POBJ_MODE_EVENT**).

The "**ErrCode**" member determines whether an error has occurred or if the transferred to "**pData**" is valid.

The **point to the data** and the **data length** are transferred in the self-made function "**GetDataInformation**".

The API "**CopyMemory**" is used in Visual Basic to copy the data to a variable. This API is declared in the declaration part of the module "**myPviFunc**".

```
Option Explicit

Public hLine As Long
Public hDevice As Long
Public hStation As Long
Public hCpu As Long
Public hTask As Long
Public hPV_Lifesign As Long

Public Declare Sub CopyMemory Lib "KERNEL32" Alias "RtlMoveMemory" _
            (lpvDest As Any, lpvSource As Any, ByVal cbCopy As Long)
```

In the "**GetDataInformation**" function, the data is now copied to the Visual Basic "**DataPV**" variable and to the "**txtLifesign**" textbox.

```
Private Sub GetDataInformation(ByVal pData As Long, _
            ByVal DataLen As Long)
Dim DataPV As Double

    CopyMemory ByVal VarPtr(DataPV), ByVal pData, DataLen
    Form1.txtLifesign.Text = DataPV
End Sub
```

Note:

In this example "**VT=f64**" is specified in the LinkDescriptor of the variable object. This results in a type conversion to "**double**" in the PVI Manager.
In the callback with data, the corresponding length is included for each data type – in this case "**8**" for the data type "**double**".

The function "**GetDataInformaton**" is implemented for simple data types in this example.

To implement the function for all data types, The "**Format Event**" can also be evaluated. This event is sent before the first data event.

More information about the creating and evaluating the format event can be found in the PVI user documentation.

Result:

After starting the program and pressing "**cmdLinkVar**" button, the value of the variable object is output to the textbox as in the exercise for the callback function without data.

3.5 Read and write access

Process objects that are switched to "**active**" by the "**EV=ed**" event mask are automatically monitored for data changes by the PVI Manager – regardless of whether these process objects are polled by the PVI Line or setup as event variables by the "**AT=re**" attribute.

That means that these process objects no longer have to be additionally read by the application.

PVI offers read and write access for accessing the respective process objects in the PVI object hierarchy.

These access functions are described in the PVI user documentation <**PVI Base System**> / <**PVICOM interface**> / <**Access types**>.

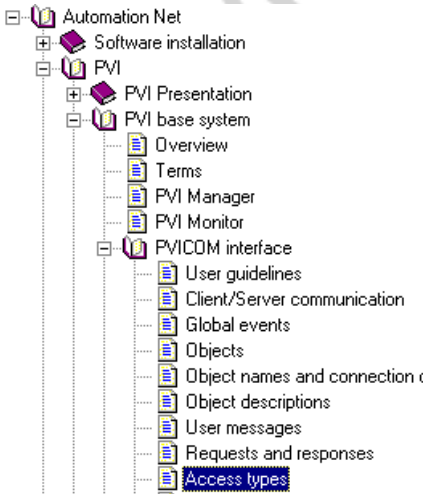


Fig. 11: Access functions

This training module uses various exercises to describe three access functions for read and write access to a process object.

3.5.1 Writing a value

The function "**PviWriteRequest()**" and the access function "**POBJ_ACC_DATA**" are used to perform asynchronous write access with data to a variable object.

The application is notified of the successful write task in the corresponding callback. Any errors that occurred are reported in the response function's return value.

Exercise:

Write to the "lifesign" variable with the value "0"

The function is implemented in the click event of a new button "**cmdWriteLifesign**". A callback function for write access should be created in the "**myPviFunc**" module.

```
Private Sub cmdWriteLifesign_Click()
Dim ReturnVal As Long
Dim WriteValue As Double

    WriteValue = 0
    ReturnVal = PviWriteRequest(hPV_Lifesign, POBJ_ACC_DATA, WriteValue, _
                Len(WriteValue), AddressOf RespWrite, SET_PVICALLBACK, 10)
    ' TODO error handling
End Sub
```

Data access to the variable object is specified by defining the LinkID (handle) of the variable object and the access type "**POBJ_ACC_DATA**".

In the callback function, the return value notifies the application whether the write access was successful (ReturnVal = 0) or if an error occurred (ReturnVal <> 0).

```
Public Sub RespWrite(ByVal WPARAM As Long, ByVal LPARAM As Long)
Dim ReturnVal As Long

    Select Case LPARAM
        Case 10:        ' Response variable object
            ReturnVal = PviWriteResponse(WPARAM)
            ' TODO error handling
        Case Else       ' Acknowledge unhandled write requests
            PviWriteResponse WPARAM
    End Select
End Sub
```

Result:

After starting the program and setting up the process objects (regardless of whiter these were created as static or temporary), the value of the variable is set to "**0**" when pressing the "**cmdWriteLifesign**" button.

3.5.2 Changing the event mask

With static process objects, the "active" switching of the process object by specifying the event mask "**EV=ed**" is controlled by the **link / unlink procedure**.

With temporary process objects, this is done by writing the process object's event mask with the access type "**POBJ_ACC_EVMASK**".

This access type can be used to enable or lock the different types of events for a process object – both static and temporary – during runtime.

Exercise:

Enabling and disabling the event types "Error" and "Data"

The event mask is "**disabled**" at index [0] and "re**-enabled**" at index [1] using an OptionButton field with the name "**optEventMask**" on the form.

These accesses can be performed using a synchronous write task because this change is made right on the PVI Manager and does not have to wait for a confirmation from the controller.

```
Private Sub optEventMask_Click(Index As Integer)
Dim ReturnVal As Long
Dim strHlp As String
Dim strLen As Long

    If Index = 0 Then        ' Eventmask "EV="
        strHlp = ""
    Else                     ' Eventmask "EV=ed"
        strHlp = "ed"
    End If
    ReturnVal = PviWrite(hPV_Lifesign, POBJ_ACC_EVMASK, ByVal strHlp, _
                Len(strHlp) + 1, 0, 0)
    ' TODO error handling
End Sub
```

Result:

When the program is started, each value change in the variable is displayed in the textbox after setting up the process objects.

The value change is no longer refreshed after pressing the OptionButton with index [0] (i.e. the variable is no longer read).

The variable is read again after "**enabling**" the event mask with "**EV=ed**".

3.5.3 Reading the controller time

In this last exercise, the controller's time will be read and output.

Note:

In addition to reading the controller's time, this function can also be used to read the time and the data from a module or a user task for service applications.
The differentiation between CPU and module objects is made by specifying the respective handle in the object hierarchy.

The access function "**POBJ_ACC_DATE_TIME**" on the **CPU object** is used to read the time from the controller.

Exercise: Read the time from the controller

Reading the time is started by pressing a new button "**cmdReadTime**". The response from the read task is displayed on a Visual Basic label control with the name "**lblTime**".

The time is saved on a structure variable in Visual Basic. The Member variables of this structure are used to perform the evaluation and formatting for the display.

The necessary time structure is created in the "**myPviFunc**" module.

```
Option Explicit

Public hLine As Long
Public hDevice As Long
Public hStation As Long
Public hCpu As Long
Public hTask As Long
Public hPV_Lifesign As Long

Private Type mDateTime
    lSecond As Long
    lMinute As Long
    lHour As Long
    lDay As Long
    lMonth As Long
    lYear As Long
    lwDay As Long
    lyDay As Long
    lisDay As Long
End Type

Public Declare Sub CopyMemory Lib "KERNEL32" Alias "RtlMoveMemory" _
            (lpvDest As Any, lpvSource As Any, ByVal cbCopy As Long)
```

The read task for the time is started in the Click event of the "**cmdReadTime**" button:

```
Private Sub cmdReadTime_Click()
Dim Returnval As Long

    Returnval = PviReadRequest(hCpu, POBJ_ACC_DATE_TIME, AddressOf _
                RespService, SET_PVICALLBACK, 1)
    ' TODO error handling

End Sub
```

A separate callback function with the name "**RespService**" is created in which the time that was read is then evaluated. An existing callback function can also be used.

Correct formatting of the display is not taken into consideration here.

```
Public Sub RespService(ByVal WPARAM As Long, ByVal LPARAM As Long)
Dim Returnval As Long
Dim tStruct As mDateTime

    Select Case LPARAM
        Case 1:          ' Response date time
            Returnval = PviReadResponse(WPARAM, tStruct, Len(tStruct))
            If Returnval Then
                Form1.lblTime.Caption = "E#" & Returnval
            Else
                ' note: +1 for month because in struct_tm
                ' the month is defined with 0-11
                ' the year starts from 1900
                Form1.lblTime.Caption = tStruct.lDay & "/" & _
                    (tStruct.lMinute + 1) & "/" & (tStruct.lYear + 1900) & _
                    " " & tStruct.lHour & ":" & tStruct.lMinute & ":" & _
                    tStruct.lSecond
            End If
        Case Else        ' Acknowledge unhandled write requests
            PviReadResponse WPARAM, 0, 0
    End Select
End Sub
```

Result:

The controller's time is displayed on the form after starting the program and pressing the "**cmdReadTime**" button.

To set the time on the controller, the time structure must be written with that of the PC and written to the controller with the "**PviWriteRequest()**" function.

4. SUMMARY

The PVI functions make it possible implement any Windows Client application, from visualizations that require data for displaying and operating up to creating Service Tools – such as the PviTransfer tool.

This training module covered a small range of the PVI functionalities. The PVI user documentation as well as the PVI Samples and the PVI Tutorial for the supported program language included in the PVI Server&Runtime / Development installation can be used to build upon basic knowledge.

The PviTutorial uses several steps to explain application of the PVI functions from creating a process object up to using most of the access functions.

Fig. 12: PVI DLL programming

Notes

Overview of training modules

TM200 – B&R Company Presentation **
TM201 – B&R Product Spectrum **
TM210 – The Basics of Automation Studio
TM211 – Automation Studio Online Communication
TM212 – Automation Target **
TM213 – Automation Runtime
TM220 – The Service Technician on the Job *
TM221 – Automation Components and Sources of Errors *
TM223 – Automation Studio Diagnostics
TM230 – Structured Software Generation
TM240 – Ladder Diagram (LAD)
TM243 – Sequential Function Chart (SFC) *
TM245 – Instruction List (IL) *
TM246 – Structured Text (ST)
TM247 – Automation Basic (AB) *
TM248 – ANSI C
TM250 – Memory Management and Data Storage
TM260 – Automation Studio Libraries I

TM400 – The Basics of Motion Control
TM402 – Dimensioning Motion Control Systems *
TM410 – The Basics of ASiM
TM440 – ASiM Basic Functions
TM441 – ASiM Multi-Axis Functions
TM445 – ACOPOS ACP10 Software
TM450 – ACOPOS Control Concept and Adjustment
TM460 – Starting up Motors *

TM600 – The Basics of Visualization
TM610 – The Basics of ASiV
TM630 – Visualization Programming Guide
TM640 – ASiV Alarm System
TM650 – ASiV Internationalization
TM660 – ASiV Remote
TM670 – ASiV Advanced

TM700 - Automation Net PVI
TM710 - PVI Communication
TM711 - PVI DLL Programming
TM712 - PVIServices
TM730 - PVI OPC

TM800 – APROL System Concept
TM801 – APROL Engineering Basics
TM810 – APROL Setup, Configuration and Recovery
TM811 – APROL Runtime System
TM812 – APROL Operator Management
TM813 – APROL XML Queries and Audit Trail
TM830 – APROL Project Engineering
TM840 – APROL Parameter Management and Recipes
TM850 – APROL Controller Configuration and INA
TM860 – APROL Library Engineering
TM865 – APROL Library Guide Book
TM870 – APROL Python Programming *
TM880 – APROL Report *

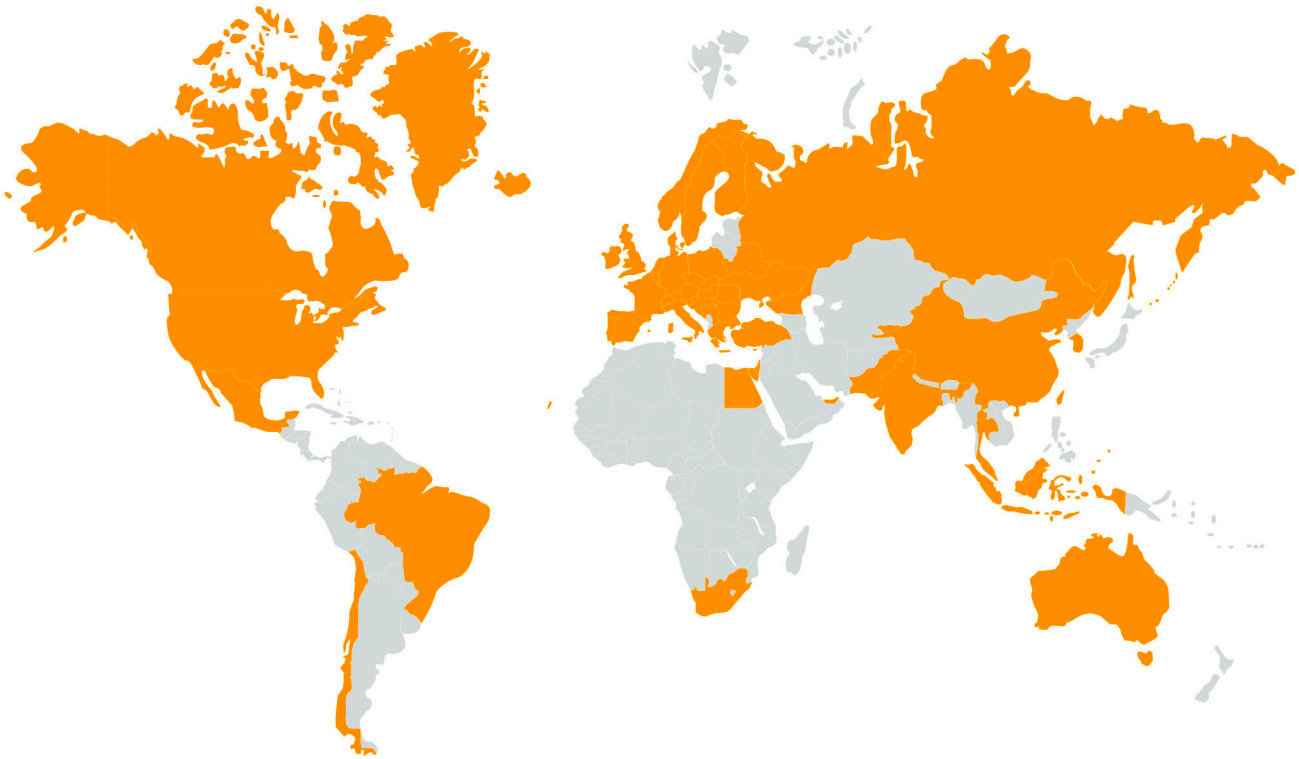
*) upon request
**) see Product Catalog

CORPORATE HEADQUARTERS

Bernecker + Rainer Industrie-Elektronik Ges.m.b.H.
B&R Straße 1
A-5142 Eggelsberg
Tel.: +43 (0) 77 48 / 65 86-0
Fax: +43 (0) 77 48 / 65 86-26
info@br-automation.com
www.br-automation.com

TM711TRE.00-ENG 0706

120 offices in more than 50 countries - **www.br-automation.com/contact**



Australia · Austria · Belarus · Belgium · Brazil · Bulgaria · Canada · Chile · China · Croatia · Cyprus · Czech Republic
Denmark · Egypt · Emirates · Finland · France · Germany · Greece · Hungary · India · Indonesia · Ireland · Israel · Italy · Korea
Kyrgyzstan · Malaysia · Mexico · The Netherlands · Norway · Pakistan · Poland · Portugal · Romania · Russia · Singapore
Slovakia · Slovenia · South Africa · Spain · Sweden · Switzerland · Taiwan · Thailand · Turkey · Ukraine · United Kingdom · USA