PVI OPC
TM730

Perfection in Automation
www.br-automation.com

Requirements

Training modules:	TM700 – Automation Net PVI
	TM710 – PVI Communication
Software:	Windows NT/2000/XP PVI Server & Runtime / Development
Hardware:	PC

Table of contents

1. INTRODUCTION 4
   1.1 Objective 5
2. PVI OPC 6
3. PVI OPC CONFIGURATOR 7
   3.1 The controller project 7
   3.2 Starting the PVI OPC configurator 7
   3.3 Components in the PVI OPC configurator 8
   3.4 Creating a new OPC database 9
   3.5 Configuring the connection to the controller 9
   3.6 Manually configuring variables 18
   3.7 Configuring variables via Online Import 22
   3.8 Data format and attributes 23
   3.9 Activating the PVI OPC database 25
   3.10 Testing the PVI OPC database 25
   3.11 OPC configurator summary 28
4. PVI OPC SERVER 29
   4.1 Connecting and setting up PVI process objects 30
   4.2 Reading and writing data 30
   4.3 Disconnecting and deleting PVI process objects 33
5. PVI OPC PROGRAMMING 34
   5.1 OPC Custom and Automation Interface 34
   5.2 Visual Basic – OPC Client 35
   5.3 Summary of OPC programming 45
6. SUMMARY 46
7. APPENDIX 47

1. INTRODUCTION

OPC (OLE for Process Control) is the standard interface – based on the DCOM component model from Microsoft – for **Windows-based SCADA packages** (**S**upervisory **C**ontrol **a**nd **D**ata **A**cquisition) to access various control systems.

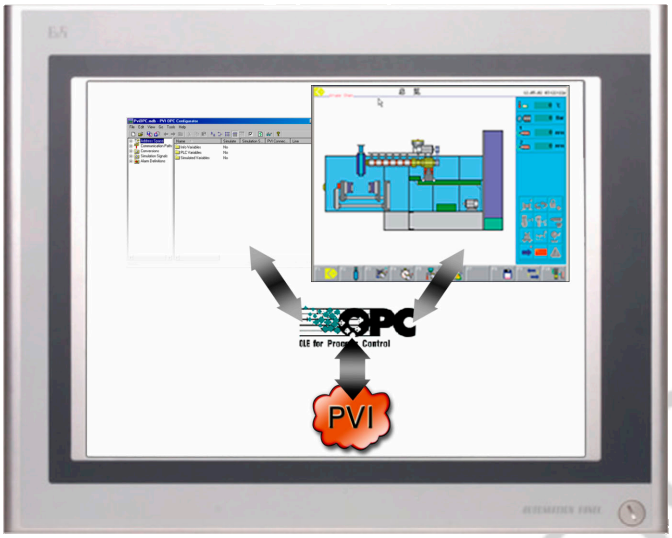


Fig. 1: PVI OPC

The OPC Server allows you to use SCADA packages from a variety of manufacturers and can also work with a number of **programming languages** including C++, Visual Basic and script languages.

This training module will explain how an OPC database is created for accessing a SCADA package and will then test it using an OPC Sample Client. This training module will also take a look at programming the OPC interface using Visual Basic / Visual Basic for Applications.

1.1 Objective

Participants will be able to create an OPC database after performing the exercises and tasks and will know the possibilities of the OPC configuration and its access possibilities.

Access to the controller via OPC can be tested using the OPC Sample Client, or any other OPC Client.

Participants will learn how to program the OPC interface by using a self-made OPC Client created in Visual Basic.

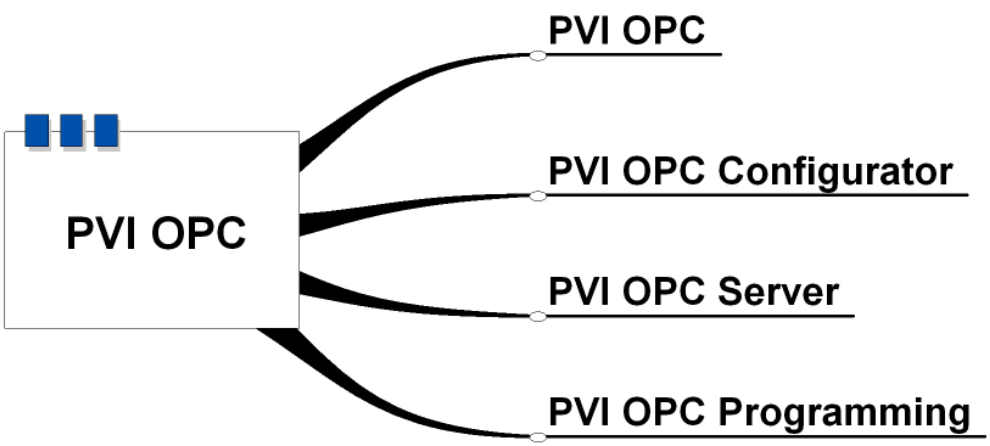


Fig. 2: Overview

2. PVI OPC

OPC (OLE for process control) is an industrial standard that was created with the participation of numerous worldwide-leading automation and hardware manufacturers in cooperation with Microsoft.

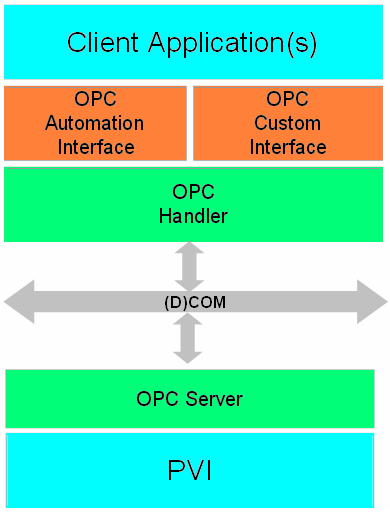
Note:

Administration and specification of the OPC interface is handled by the **OPC Foundation**. This foundation is made up of members from well-known visualization and controller manufacturers.

www.opcfoundation.org

B&R uses a toolkit for creating the PVI OPC Server, which contains the standard interface for accessing SCADA packages or for programming.

OPC is based on Microsoft's OLE (Object Linking and Embedding) and DCOM (Distributed Component Object Model) technology and comprises a set of standard interfaces, features and methods used by automation clients for process control and manufacturing.



The OLE/COM technologies determine how individual software components work together and exchange data. OPC provides a general interface for communication with various process control devices, independent of the control software used in the process.

The standardized interface enables the user to select any SCADA package that supports OPC or to create his own OPC Client based on VC++ or VB.

3. PVI OPC CONFIGURATOR

The PVI OPC Configurator is contained in the PVI Server & Runtime / Development package. The configurator can only be installed together with the OPC Server and its components.

3.1 The controller project

A controller project is needed in order to test the OPC configuration.

Any existing control application can be used for this. In this training module we will be accessing an AR000 project with the following variables:

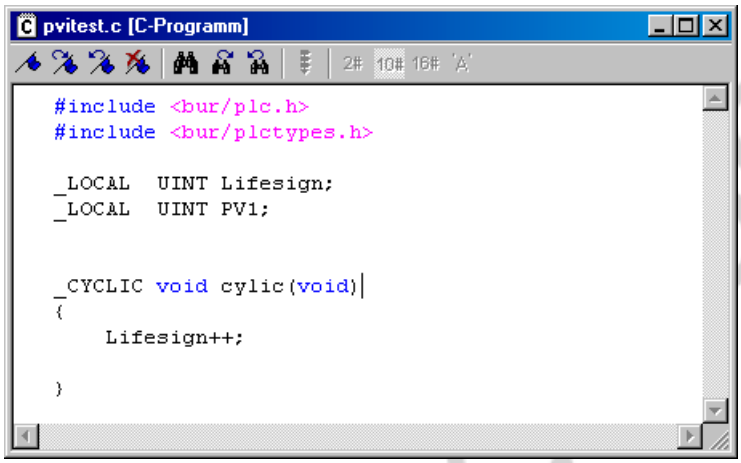
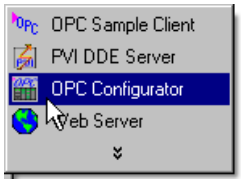


Fig. 3: Automation Studio task "pvitest"

This task will be expanded as necessary in the individual exercises.

3.2 Starting the PVI OPC configurator

The OPC configurator is started via <**Start**> / <**All Programs**> / <**B&R Automation**> / <**Server**> / **OPC Configurator**.



3.3 Components in the PVI OPC configurator

A sample database is loaded when opening the OPC configurator for the first time (PviOPC.mdb).

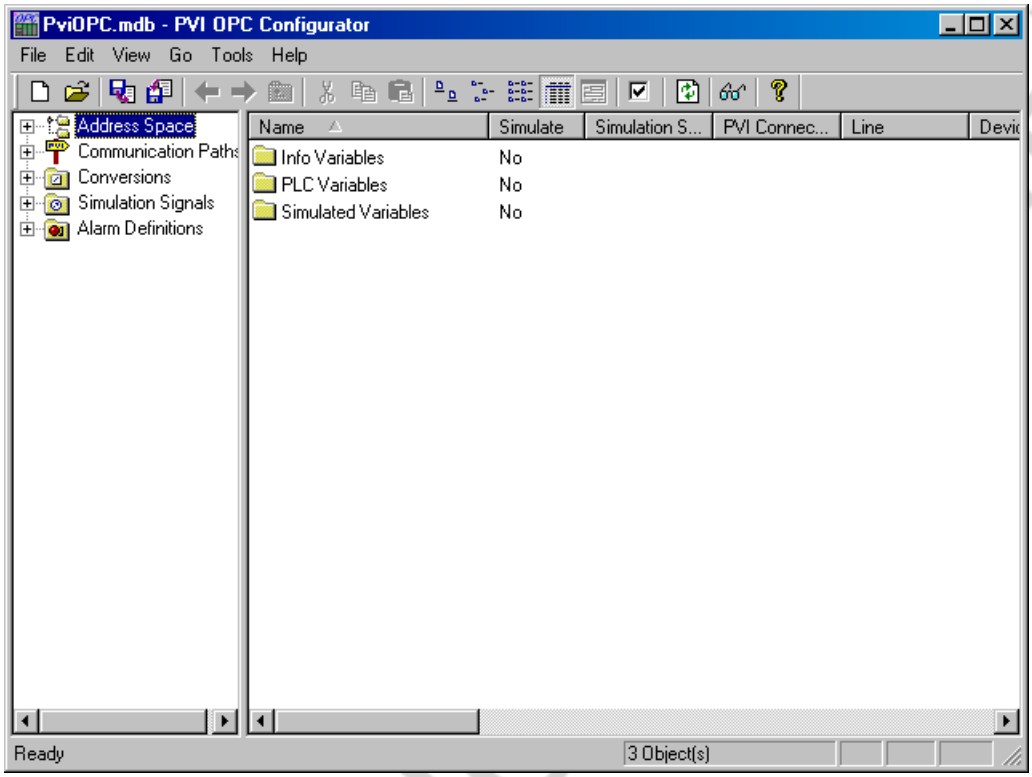


Fig. 4: PVI OPC configurator – Default view

The PVI OPC configuration is made up of the following parts:

- Address Space
- Communication Paths
- Conversions
- Simulation Signals
- Alarm Definitions

All control variables (also called **Items**) are managed in logical groups (folders) in the **Address Space**.

The controller connection up to the task object is configured in the **Communication Path** using configuration dialog boxes.

The component **Conversions**, **Simulation Signals** and **Alarm Definitions** will not be covered explicitly in this training module. Further information can be found in the PVI user documentation "*PVI Server/PVI OPC*".

3.4 Creating a new OPC database

A new OPC database will be created for the following exercises. This is created by selecting <**File**> / <**New**> or by clicking on the "**New**" button .

Now specify a directory and the file name **TM730.mdb** in the **Save** dialog box that appears.

A new – blank – database is then created.

3.5 Configuring the connection to the controller

The first step when creating the configuration is to create the Communication Paths so that a connection can be established with the controller.

During development:

- Online import of variables
- Testing the configuration in the Monitor view

During runtime:

- Connection to the controller and registering the variables

Exercise:

Creating the communication path

A new **PVI connection** is added using the shortcut menu (right-click) in the "Communication Path".

Note:

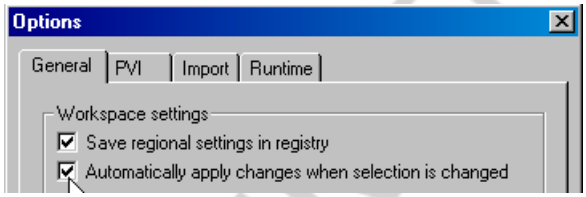
The configuration must be adjusted accordingly depending on the media being used to connect to the controller (TCPIP, serial, etc).
In this exercise, the connection to the AR000 is established via TCPIP.

The connection description of the corresponding PVI process object is defined in the PVI parameter dialog boxes.

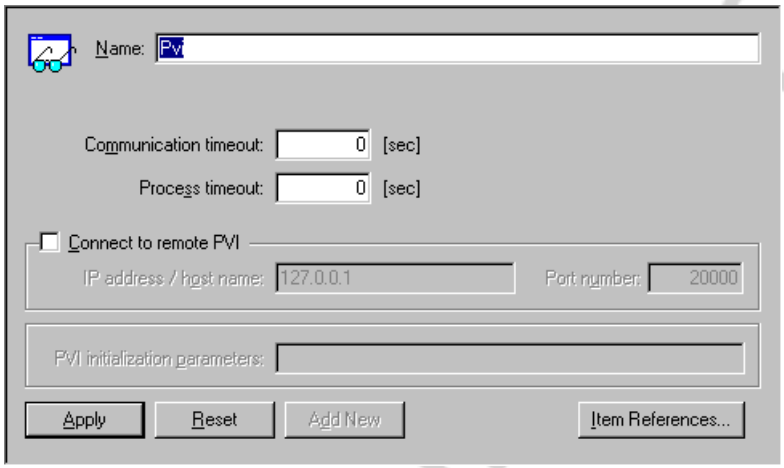
Note:

All changes must be confirmed by clicking the <**Apply**> button. The next PVI object cannot be added until this has been done.

Experienced users have the option to deactivate this behavior. This setting can be made via <**Tools**> / <**Options**>.



3.5.1 PVI object



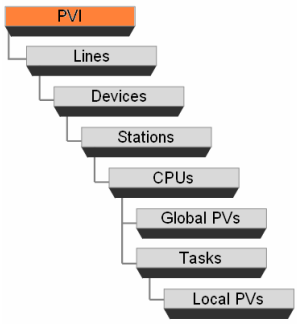


Fig. 5: PVI process object

A PVI object is added to the **Communication Paths** folder after completing the configuration by selecting <**Apply**>.

3.5.2 Line object

The line must now be selected from the shortcut menu of the PVI object added above.

No changes are necessary here because the INA2000 line is already shown as the default line.

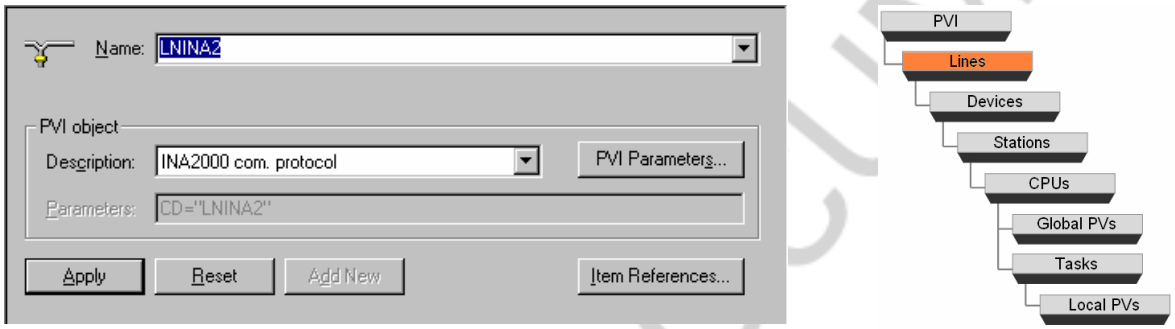


Fig. 6: Line process object

A line object with the name of the line **LNINA2** is added to the **Communication Paths** folder under the PVI object after completing the configuration by selecting <**Apply**>.

Caution:

The connection description is displayed in the "**Parameters**" line. Experienced users can also enter the connection description manually. This option is enabled via the menu <**Tools**> / <**Options**> / <**PVI**> / "**Enable direct PVI parameter input**".
However, we recommend always using the <**PVI Parameters**> dialog box to make the settings.

3.5.3 Device object

The medium (i.e. the device) must now be selected from the shortcut menu of the line object and configured.

The Ethernet interface is selected by default. The connection to the AR000 is made via the TCPIP medium.

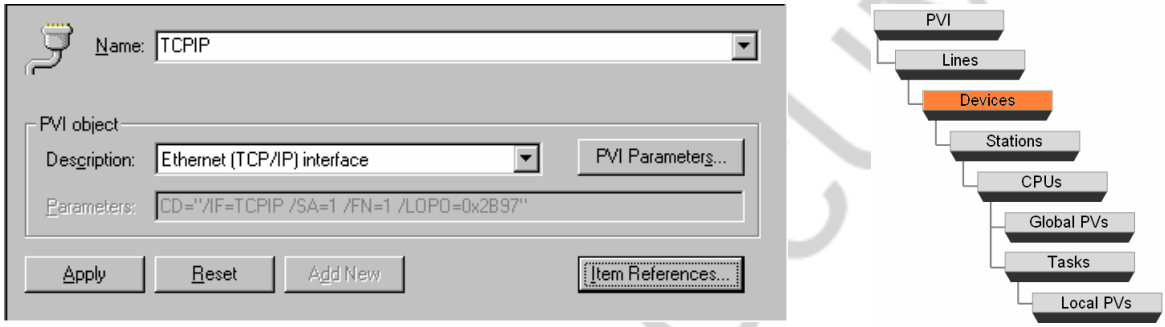


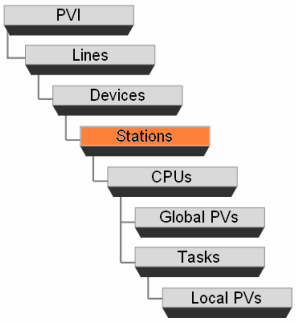
Fig. 7: Device process object

A device object with the name of the device **TCPIP** is added to the **Communication Paths** folder under the line object after completing the configuration by selecting <**Apply**>.

3.5.4 Station object

A station must now be added using the shortcut menu of the device object. This object does not have to be configured.

A station object with the name **Station** is added to the **Communication Paths** folder under the device object after completing the configuration by selecting <**Apply**>.



3.5.5 CPU object

A CPU must now be added using the shortcut menu of the station object.

This object must be configured for the AR000 communication.

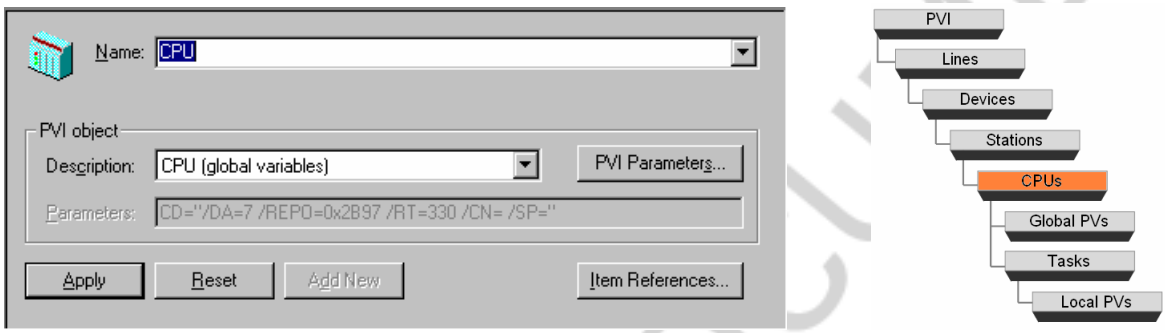


Fig. 8: CPU process object

The configuration dialog box for the CPU object is opened by clicking the button <**PVI Parameters**>.

The following entries must be changed in this dialog box:

- 1: Destination Address = 2
- 2: Remote Port Number = 2B98
- 3: IP Address = 127.0.0.1

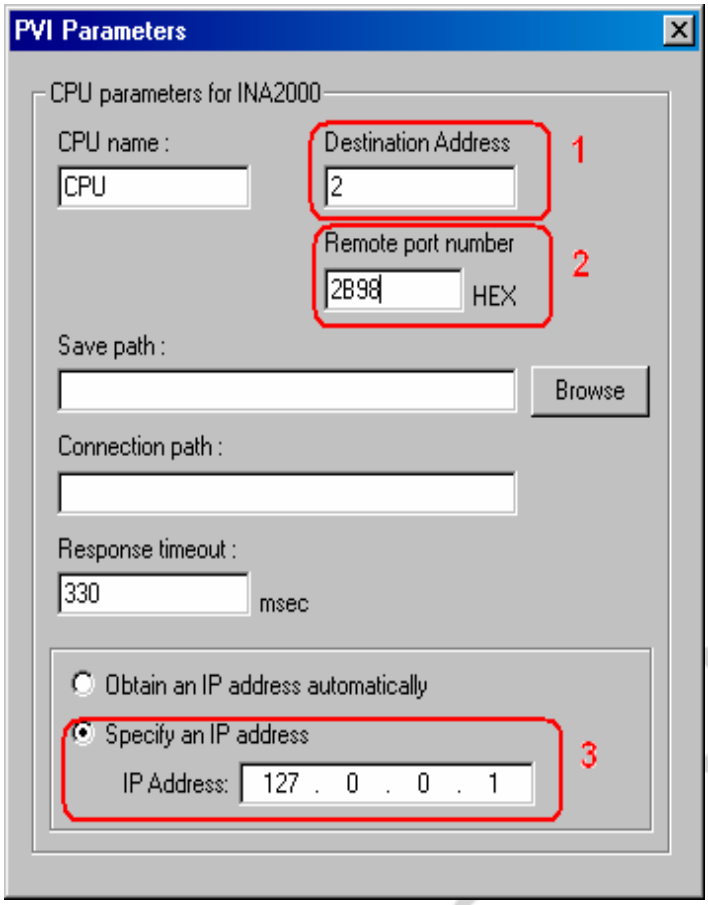
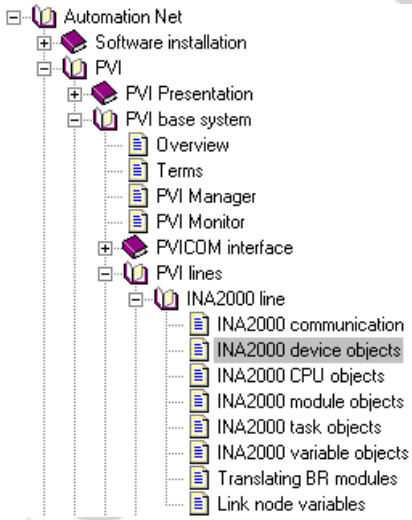


Fig. 9: PVI Linien Dokumentation



The parameters in the configuration dialog box depend on the medium being used.

A description of this parameter can be found in the documentation for the PVI INA2000 line.

A CPU object with the name **CPU** is added to the **Communication Paths** folder under the station object after completing the configuration by selecting <**Apply**>.

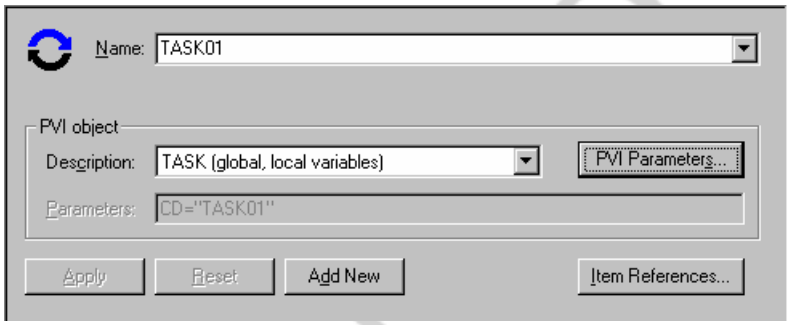
3.5.6 Manually adding a task object

A task object is created as the last PVI process object.

> **Note:**
>
> A task object is not necessary if only global variables are being used. However, we also recommend registering all global variables in one task object because this can speed up the variable registration using the save path (configuration on the CPU object).

The task object is created in the shortcut menu of the CPU object.

The task name of the control project that the OPC variables will refer to must be entered in the task object.



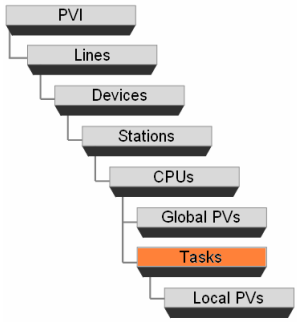
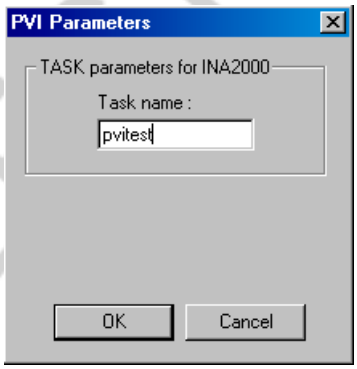
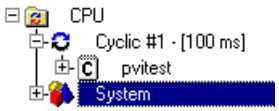


Fig. 10: Task process object

The configuration dialog box for the task object is opened by clicking the button <**PVI Parameters**>.

The name of the Automation Studio task in this practice example is "**pvitest**".





In the next step, the user is asked whether the task name should also be used for the name of the task object. This step is confirmed by selecting <**Yes**>.

The connection to the controller – in our case the AR000 – is now configured.

The **Communication Paths** folder should now look like this:

3.5.7 Adding task objects via Online Import

The task objects can be added via "**Online Import**" once the CPU object has been defined and a connection to the controller has been established.

Exercise:

Using the online import

Delete the **pvitest** task object and use the **Online Import** function. The same structure should then appear in the **Communication Paths** folder as with the manual configuration of the task object.

The Online Import function is opened from the shortcut menu of the CPU object:

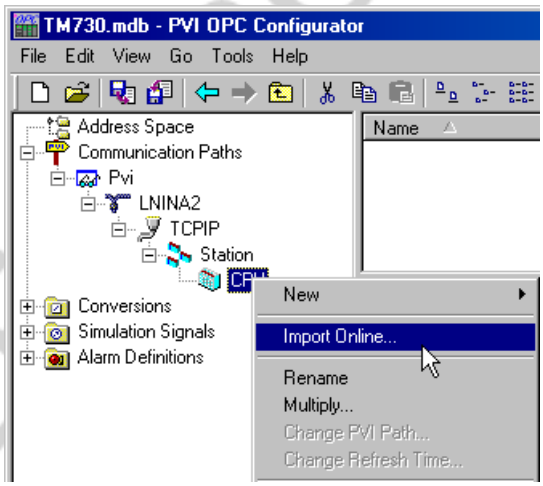


Fig. 11: Online Import for task objects

All of the tasks in the controller project are displayed in the next dialog box after the connection to the controller has been successfully established.

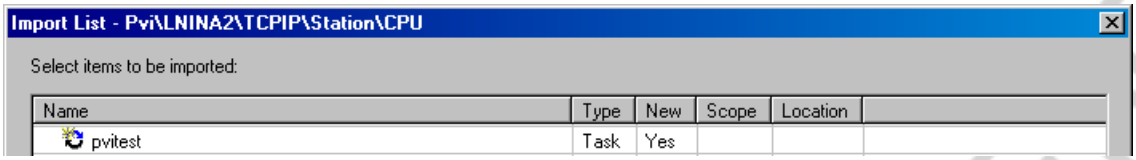
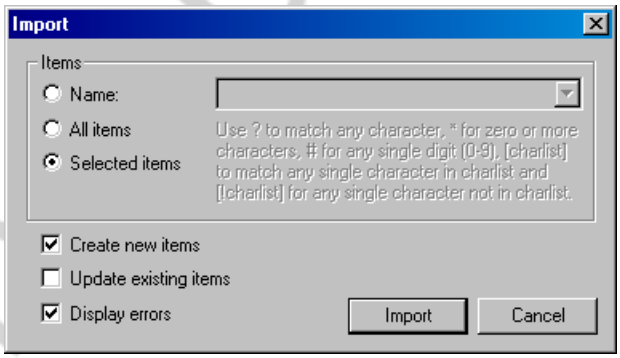


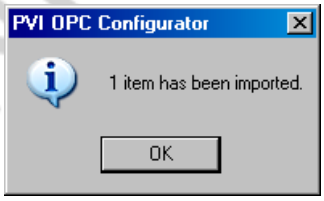
Fig. 12: Online Import of the task list

After selecting the "**pvitest**" task and pressing the <**Import**> button, the user is queried in the next dialog box whether all of the tasks or only the selected tasks should be imported.

The import procedure is started with the default settings by pressing the <**Import**> button.



The following window is displayed after successfully importing the selected task. Click <**OK**> to continue.



The task object "**pvitest**" is placed under the CPU object.

3.6 Manually configuring variables

The **data items** (i.e. the control variables), which are required for accessing the OPC Client during runtime, are configured in the **Address Space** folder.

These variables can either be configured manually or via the **Online Import**.

To do this, each of these **data items** must be linked to a PVI object (CPU or task).

> **Note:**
>
> To manage the variables in a more clear and organized manner, we recommend distributing them in logical folders instead of placing all of the variables right in the **Address Space** folder.

All **data items** are managed in folders.

3.6.1 Creating a new folder

A new folder is added by selecting <**New**> / <**Folder**> from the shortcut menu of the **Address Space** or by using the shortcut <**CTRL**> + "**F**".

Fig. 13: Creating new folders

A folder can be added to any level of an existing folder.

Task:

Create a new logical folder with the name:
"PLC Variables"

3.6.2 Adding a data item

A **data item** with a reference to a controller variable in the **pvitest** user task is created under the newly created folder.

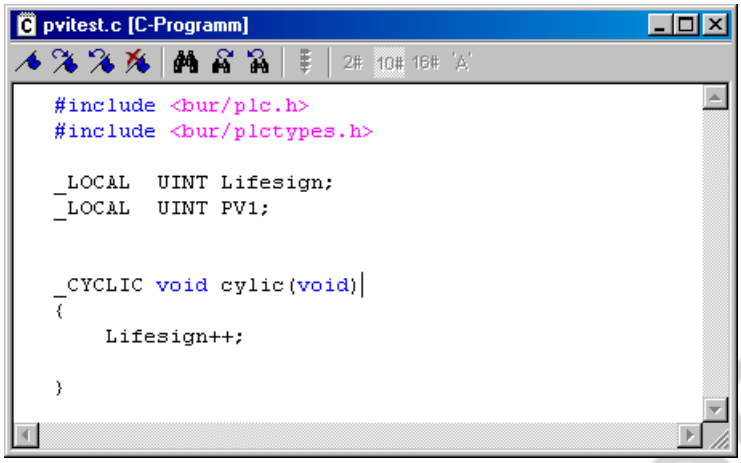


Fig. 14: "pvitest" user task

A new **data item** is added by using the shortcut menu of the folder or by using the shortcut <**CTRL**> + "**J**".

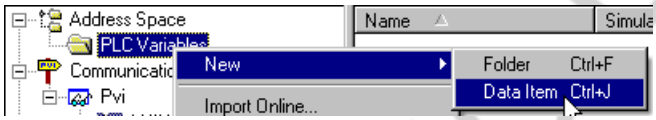
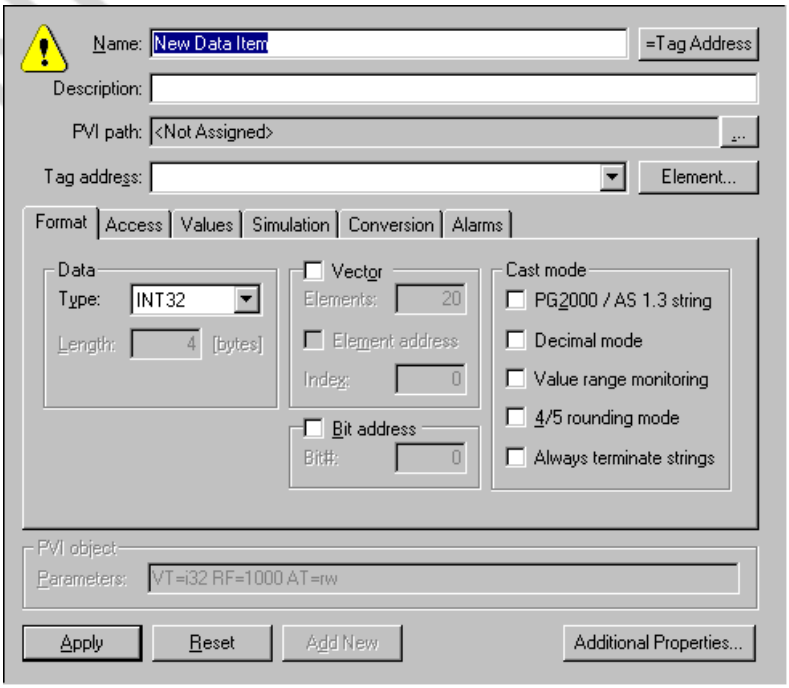


Fig. 15: Adding a new data item

The configuration dialog box for the **data item** is shown on the right side.



The **data item** is configured in the following steps:

The **Name** line specifies the **symbolic item name**. The connection to the OPC Client is made using this name.

The controller's **physical variable name** is entered in the **Tag Address** line.

Variable addresses are not limited to single variables. Elements of a structure and elements of an array can also be entered.

For example: Structure.Element
Structure[0].Element[0]
Element[0]

Note:

The symbolic name is used as variable name if the Tag Address line is left empty.

The variable is linked to a CPU or task object by selecting the corresponding **PVI path**.

All configured PVI objects are displayed by pressing the <**Selection**> button  .

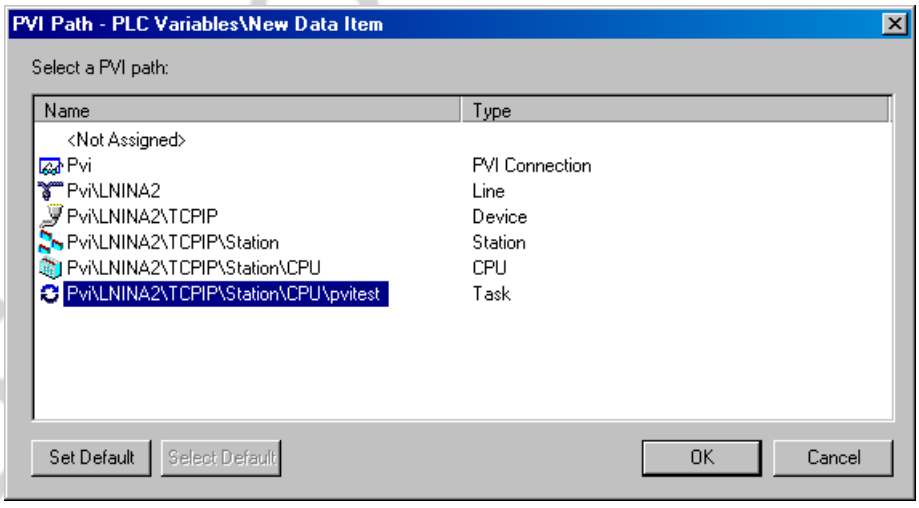


Fig. 16: Selecting the PVI object

All new subsequent **data items** are immediately linked with the corresponding PVI object by clicking the <**Set Default**> button.

If there is a valid online connection to the controller, then the **Tag Address** can be selected from the combo box after the **PVI path** has been defined.

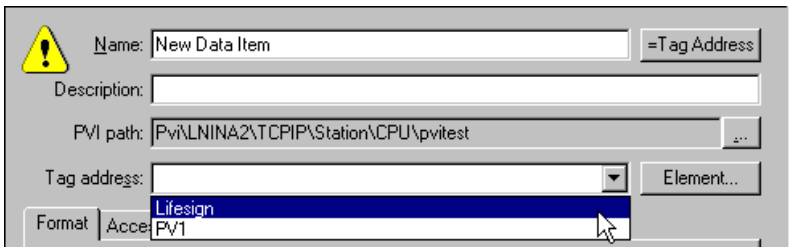


Fig. 17: Selection of the Tag Address with online connection

Task:

Add the two variables contained in the control task.

Add the two variables configured in the pvitest task to the OPC configuration.

The symbolic names of the variables in the "**Name**" line should describe the variable's function.

There should now be two **data items** in the **PLC Variables** folder.

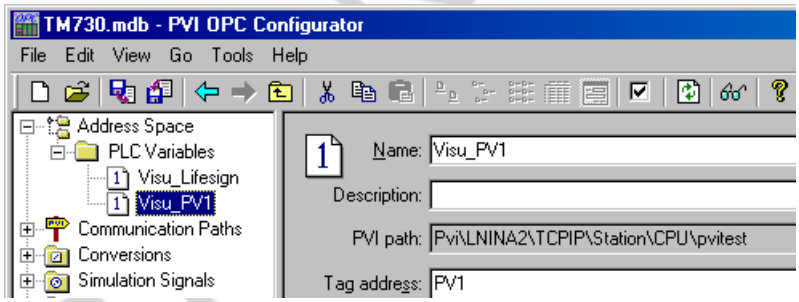


Fig. 18: Configured data item

3.7 Configuring variables via Online Import

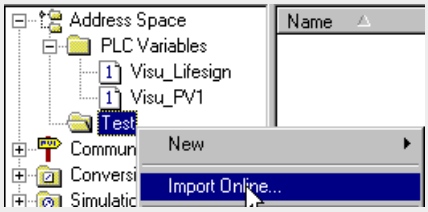
Similar to importing a task, variables can also be imported from the controller online.

Imported variables or data items are placed in the folder where the cursor is located at the time of importing.

Exercise:

Online import of variables

Create a new folder called "Test". The online import function can be started using the shortcut menu in this folder.



The OPC configurator attempts to establish a connection with the controller after the task object has been selected from the next dialog box.

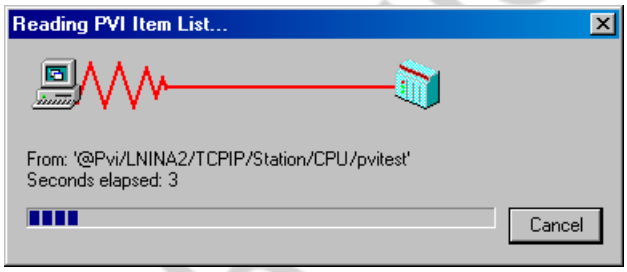


Fig. 19: Reading the variable list

The variables of the selected task, **pvitest**, are displayed once the connection has been established. Highlight the desired variables, press the <**Import**> button and confirm your selection in the subsequent window to import the variables.

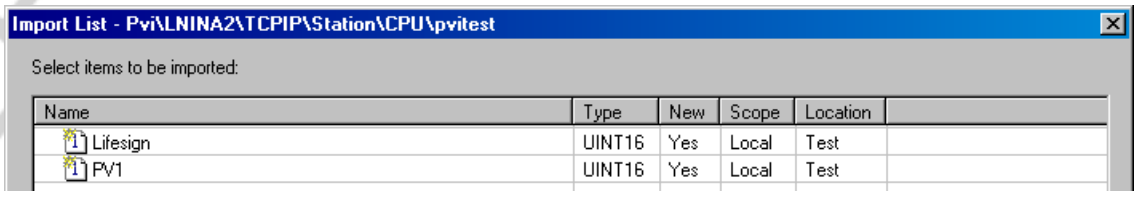
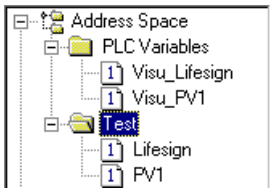


Fig. 20: Dialog box for importing variables

The imported variables are shown in the **Test** folder with the item name of the controller variable (tag address).



Note:

The symbolic variable name might have to be manually edited when importing variables.

3.8 Data format and attributes

The **format** defines the **data type**, the configuration of arrays (= **vector**) and the **Cast Mode** of the data item.

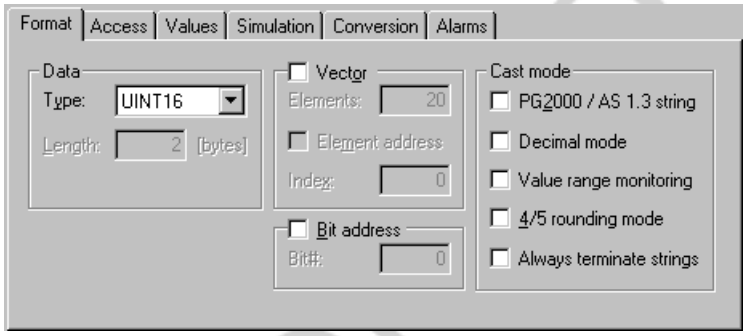


Fig. 21: Data item format

Information for the individual entries can be found in the PVI OPC user documentation.

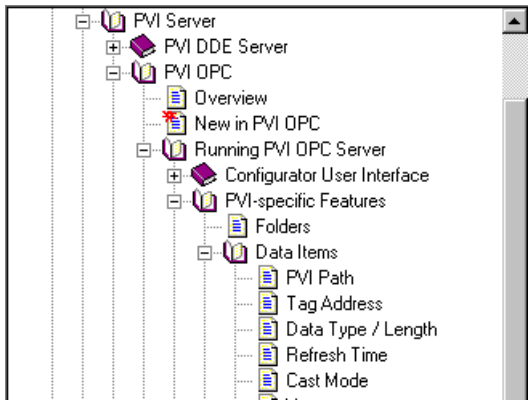


Fig. 22: PVI OPC – Data item documentation

The attributes are defined in the **Access** folder and define the access rights, the setting for the event-controlled communication and the refresh time.

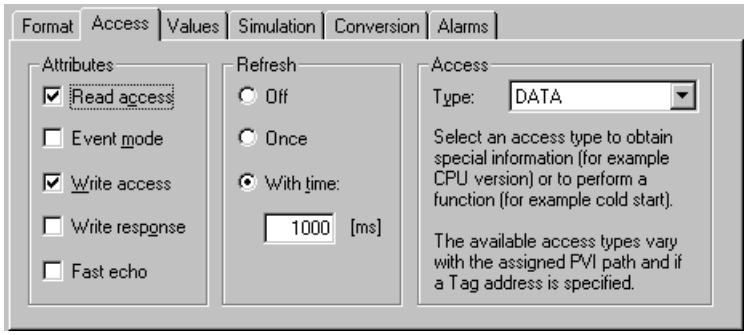


Fig. 23: Data item - Access

Only the attributes that are important for communication performance are described in this training module. All other attributes can be found in the PVI OPC user documentation.

The **event mode** and the **refresh time** can considerably influence communication performance.

Caution:

The defined **refresh time** is not a guarantee for the variable update in the OPC Client because the OPC Client reads the data from the OPC Cache using a predefined time.

Information about the OPC Client's defined update time can be found in the documentation of the software being used.

A detailed description of the **event mode** and the **refresh time** can be found in the TM710 training module.

The **read access** and **write access** attributes define whether read or write access is allowed by the OPC Client.

The **write response** attribute defines whether the program will wait for a response from the OPC Server during a write task from the OPC Client (controller response). A response to the write task is immediately sent to the OPC Client if this option is deactivated.

3.9 Activating the PVI OPC database

The database should be activated if the PVI OPC configurator is closed or the Monitor View is activated in the configurator.

The OPC Server uses the active database (reference to *.mdb file) during runtime.

The database can also be activated via <**File**> / "**Make active...**".

The OPC Client can then access the active database once the dialog box has been confirmed by clicking <**Yes**>.

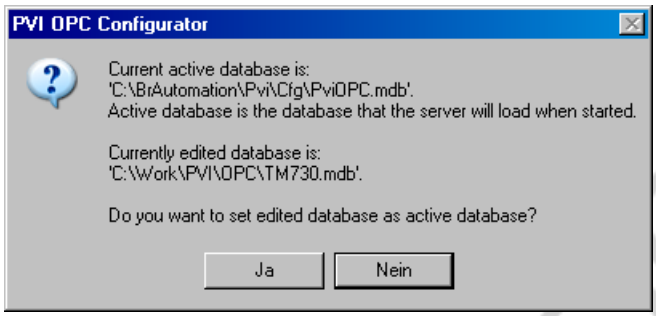


Fig. 24: Activating the database

3.10 Testing the PVI OPC database

There are two ways to test the new database:

- Monitor View of PVI OPC configurator
- OPC Sample Client

3.10.1 Testing with the Monitor View (read only)

The PVI OPC configurator can be used to check whether or not a **connection** can be established with the controller and to read values from the configured **data item**.

> **Note:**
>
> Values can only be displayed in the Monitor View. It is not possible to write to data items.

If a folder is selected in the **Address Space**, then the Monitor View can be activated by pressing the **Monitor View** button.

All **data items** are shown in the lower part of the PVI OPC configurator with their values including the **time stamp** of the data change and the **quality**.

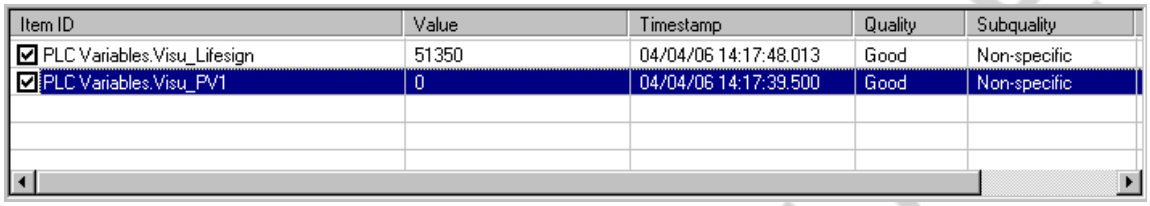


Fig. 25: OPC Monitor View

Appearance when the connection to the controller is not working:

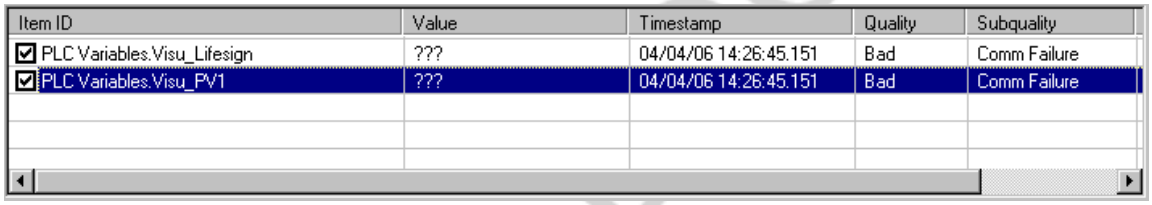
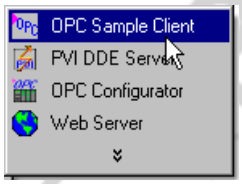


Fig. 26: OPC Monitor View with connection error

3.10.2 Testing using the OPC Sample Client (read and write)

When installing the OPC Server & configurator, an **OPC Sample Client** is also automatically installed.

It is started via <**Start**> / <**All Programs**> / <**B&R Automation**> / <**Server**> / **OPC Sample Client**.



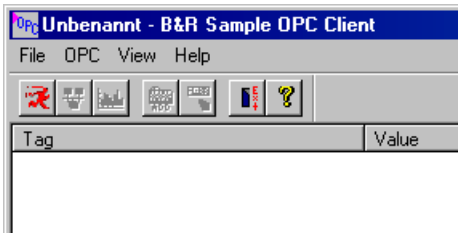


Fig. 27: OPC Sample Client

All of the OPC Servers registered in the system are shown by pressing the <**Connect**> button  .

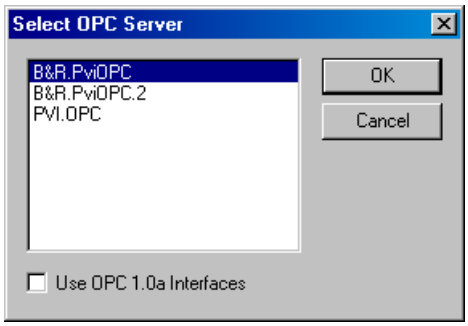


Fig. 28: Connecting the OPC Server

Either the OPC Server **B&R.PviOPC** or **B&R.PviOPC.2** can be selected. At the moment, there is not difference between the two.

Individual variables from the configuration can be selected from the Add Item dialog box by pressing the <**Add**> button  .

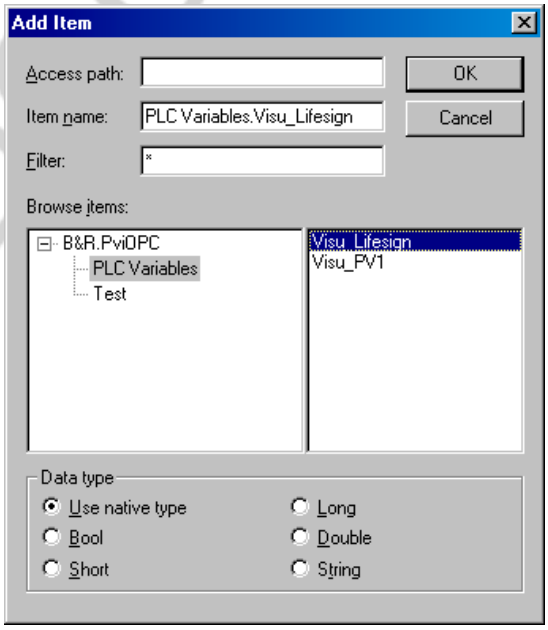


Fig. 29: Add Item dialog box

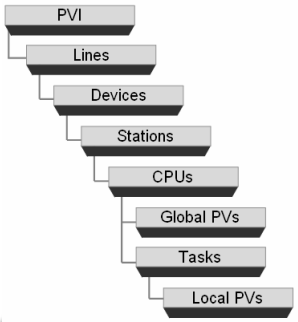
The highlighted variable object is selected by clicking <**OK**>. The current value is also displayed if a connection already exists.

Tag	Value
PLC Variables.Visu_Lifesign	23317

A write procedure is possible by pressing the <**Write Item**> button  and entering the desired value.

3.11 OPC configurator summary

The PVI objects are configured in the **Configuration Path** according to the **object hierarchy**. The connection description of the process objects is made in parameter dialog boxes.



All **data items** are put into logical groups in the **Address Space** and normally connected to a PVI task or CPU object in the **Configuration Path**.

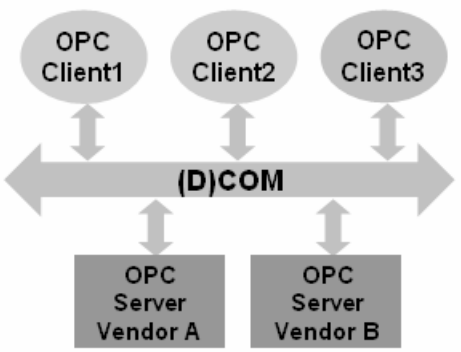
> **Note:**
>
> The data items are connected directly to the device object in the MTC and ADI lines.

The task objects in the **Configuration Path** and the **data items** can be configured via **online import** if an online connection is available. In this case, only one connection to the CPU object has to be configured.

The configuration can be tested using the **Monitor View** or the **OPC Sample Client** if an online connection is available. An additional client is not necessary.

4. PVI OPC SERVER

The B&R PVI OPC Server is an OPC-compatible server that can communicate with various I/O devices and protocols as well as send data to OPC Clients when using the B&R Process Visualization Interface (PVI).



A connection to the OPC Server is already provided when setting up a project with the PVI OPC configurator. The **data items** in the database can already be accessed at this point.

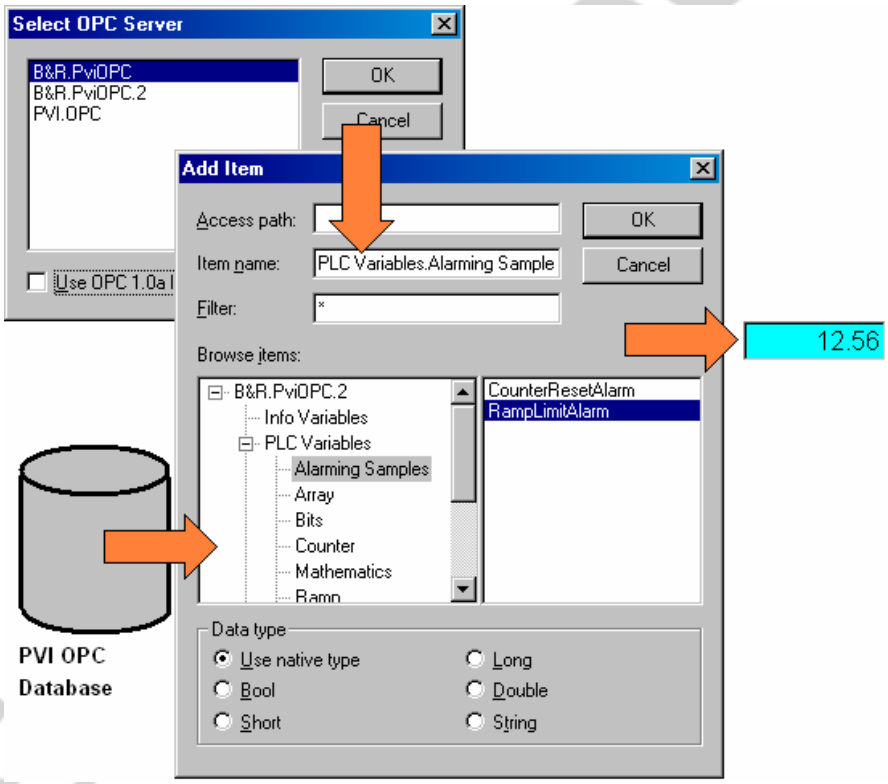


Fig. 30: B&R PVI OPC Server

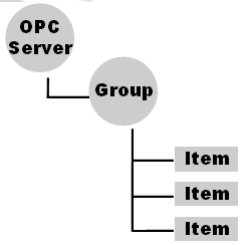
Note:

The PVI OPC server supports OPC Data Access Version 1.0 and 2.0 as well as OPC Alarms & Events Version 1.0 and can be used with DCOM for Intranet and Internet applications.

4.1 Connecting and setting up PVI process objects

When a **data item** is registered for the first time (added to a group), the PVI OPC Server attempts to connect the corresponding **PVI process object**.

If the connection fails, then the server attempts to set up static PVI process objects that have been assigned to data items for each item in the PVI path and the Tag Address, before trying to connect the PVI process object again.



Each process object must be "identified" the first time a connection is established. This increases the time until the first data change in the OPC Client.

By default, the data items are only adjusted the first time they are used. For example, when a page is changed on the OPC Client, the data items connected to the new screen page will be registered again.

This behavior can be changed in the runtime system's OPC configurator. All process objects could then be set up when starting the server – select <**Tools**> / <**Options**> / <**Runtime**> / "**Create PVI process objects at start**".

All process objects are set up as static. That means that these will remain registered even when the OPC Client is ended on the PVI Manager.

Caution:

Changes made to the connection description (Tag Address, Format, Attribute) in the OPC configuration are not effective until the PVI Manager has been restarted.

4.2 Reading and writing data

The server performs a synchronous read task after a data item is registered for the first time in order to acquire the item's data for the first time.

Changes made to the data in a process object are sent from the PVI to the server via callbacks and are then saved in a **CACHE** by the server.

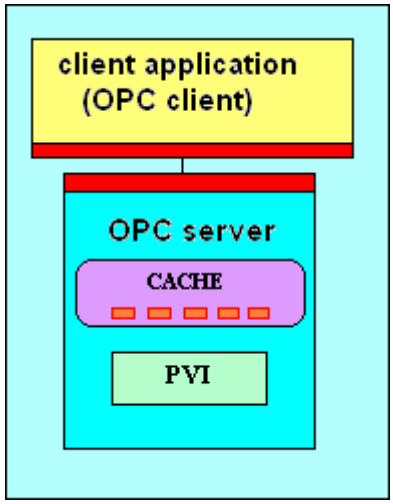


Fig. 31: OPC cache

The **CACHE** is updated according to the refresh time defined in the PVI OPC configuration. The data stored in the **CACHE** is sent to the client either during a client read task (read from cache) or via a data event with an update rate defined by the client.

4.2.1 Data item properties

Every **data item** is represented by **three** properties:

- Value: Numeric or textual
- Time stamp: Time stamp of the data item's last value change
- Quality: Quality of the data item

Caution:

The time stamp is generated by the OPC Server as soon as a data change event is received. The time does not correspond to the value change on the controller!

4.2.2 Read from cache / Read from device

There are two possibilities for acquiring the data (read tasks) using the OPC Client.

The OPC Client reads the data from this cache when using the "**Read from cache**" access type.

However, this also means that the read access occurs asynchronous to the data acquisition and the update of the cache via the PVI.

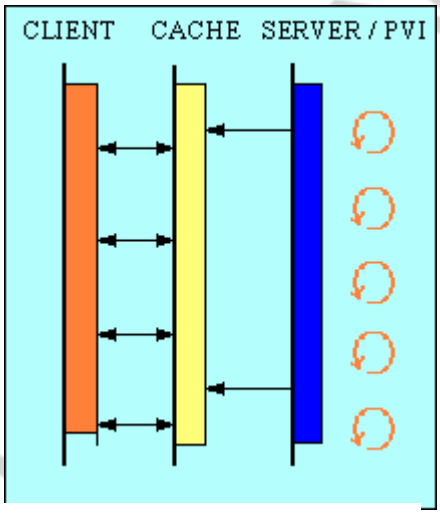


Fig. 32: Read from cache

The "**Read from device**" access type leads to a synchronous PVI read task in the PVI OPC Server. This type of access causes the application to "**freeze**" in the OPC Client until the response data has been read from the controller.

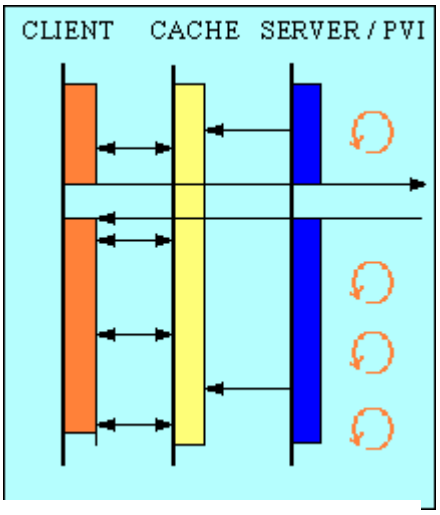


Fig. 33: Read from device

Caution:

This type of access should only be used when synchronous access to controller variables is necessary. Otherwise, the access type "read from cache" should be used.

4.2.3 Writing values

If the OPC Client sends a write task to the controller, then it is processed according to the "Write response" data item attribute.

The write task is ended with "Write response" if a response was received from the controller.

Without "Write response", the write task will be ended right after the data has been sent.

4.3 Disconnecting and deleting PVI process objects

The server maintains the connection to the process object throughout the lifetime of the data item. That means that the connection to the process object is broken by the server (inactive process object) once the data item is no longer being used by a client.

The process object is activated again the next time the data item is registered on the PVI OPC.

This behavior can be changed in the runtime system's OPC configurator. All process objects could then be set up when ending the server – select <**Tools**> / <**Options**> / <**Runtime**> / "**Delete PVI process objects at end**".

5. PVI OPC PROGRAMMING

When accessing the OPC Server from a programming language such as C++, Visual Basic or a Script language, there are two types of interfaces in the OPC specification; the **OPC Custom interface** and the **OPC Automation interface**.

5.1 OPC Custom and Automation Interface

Custom interfaces are used in programming languages such as Visual C / C++.

Programming languages such as Visual Basic or Visual Basic for Applications do not support function pointers. The **Automation interface** was introduced so that OPC Client applications could also access DCOM objects.

The Automation interface uses a standard interface which calls methods by their names instead of using function pointers.

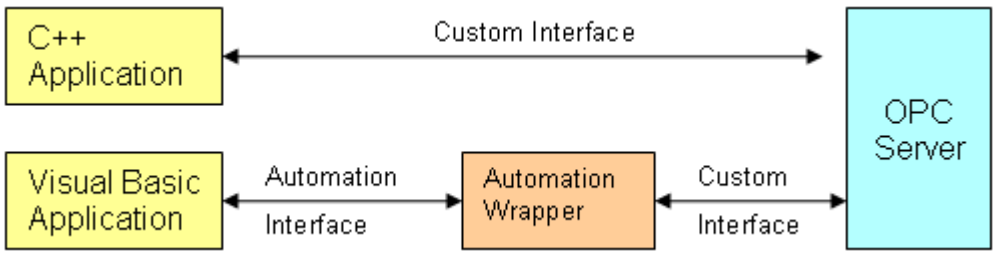


Fig. 34: Automation interface

An automation object provides features that make it possible to read or set the static properties of an object.

The OPC client is notified of changes via events.

The Automation interface functions are provided in an Automation wrapper type library (DLL).

5.2 Visual Basic – OPC Client

Access via the Automation interface for data access to the OPC Server is described using a small Visual Basic application.

The following image shows the object model provided by the Automation wrapper.

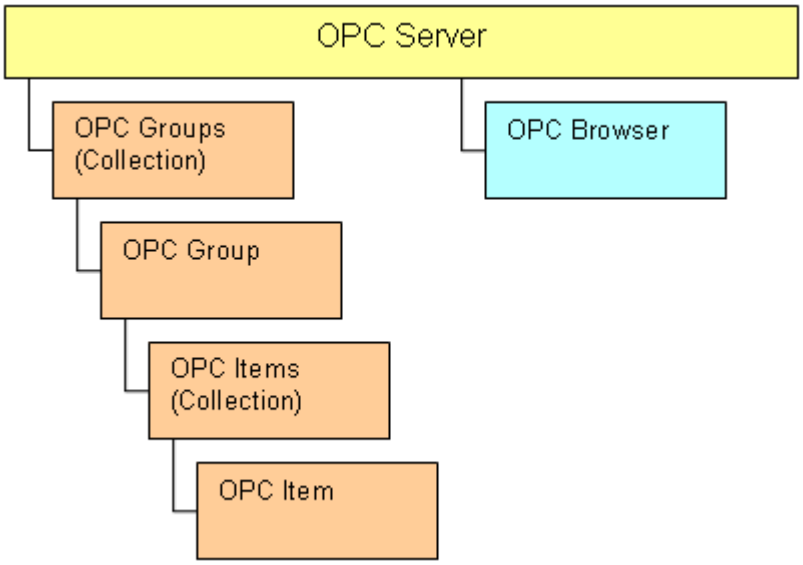


Fig. 35: Automation wrapper

Note:

Programming a Visual Basic program is not a requirement for this example. Even Visual Basic beginners can create a program capable of running by following the individual steps.

The OPC database created in section 3 is used for accessing the data items.

5.2.1 Starting Visual Basic and creating a VB application

Exercise:

Starting Visual Basic and creating a new standard *.EXE program.

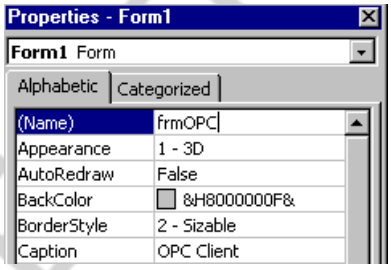
A new executable program is created after starting Visual Basic 6.0.

The VB interface appears after clicking the <**Open**> button and a new "**Form**" screen is opened.

Note:

It is recommended to give each component a meaningful name. This gives programs a more clear and organized overview.

The name and the caption are changed in the properties of the "**Form1**" screen according to the subsequent page.



5.2.2 Integrating the OPC Automation wrapper library

The Automation wrapper DLL must be integrated in the project before the VB components can be accessed or programming is possible on the OPC Server.

This is done using the menu <**Project**> / <**References**>. Select the **OPC Automation 2.0** object from this dialog box.

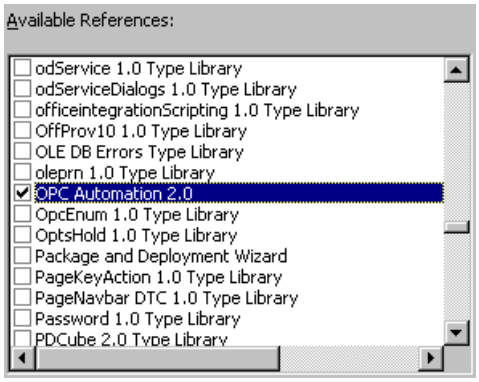


Fig. 36: OPC Automation wrapper library

The following classes are available after integrating the OPC Automation object:

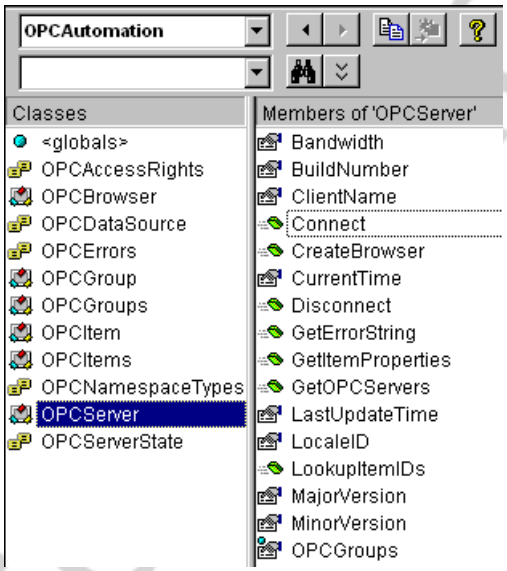


Fig. 37: PC automation class overview

This overview can be opened by pressing the <**F2**> key.

5.2.3 Variable declaration

The necessary variables must be declared before the objects can be accessed.

Double-clicking on the form opens the Program Editor for entering the following program code:

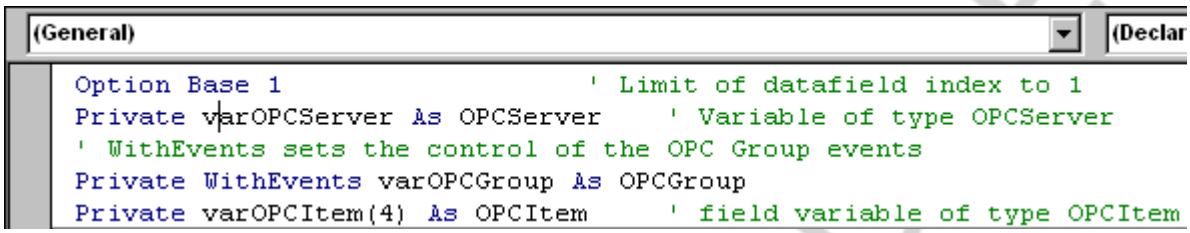


Fig. 38: Variable declaration for OPC access

5.2.4 Visual Basic objects on the form

Two **CommandButton** objects with the name **cmdConnectOPC** and caption **Connect** and the name **cmdDisconnectOPC** and caption **Disconnect** are placed.

Two TextBox controls are also required for displaying and entering values. These are given the names **txtVisuLifesign** and **txtVisuPV1**.

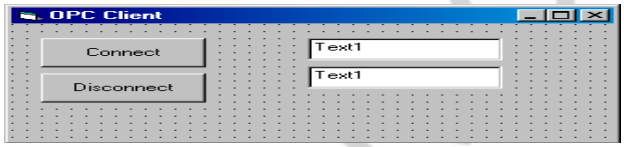


Fig. 39: Objects in the Visual Basic project

Caution:

The "**Enabled**" property for the **disconnect** button should be set to **false**. Otherwise, a program exception occurs if the button is pressed while executing the program before connecting.

5.2.5 Connect function

The first access to the OPC Server writes the **Connect**, (i.e. establishes the connection).

Programming is possible by double-clicking on the <**Connect**> button.

The following procedure occurs with **Connect**:

- Declare variables for the collection objects
- Create a new OPC Server Automation object
- Call the **Connect** method for this object. The Wrapper starts the Data Access Server with custom interfaces
- Output the VendorInfo (i.e. the OPC Server name) to the output window
- Create an OPCGroup object and add a group with the name **OPCSample** via the **Add** method
- Set the refresh time (UpdateRate) and the status
- Enable the callback connection

The following program code should be entered:

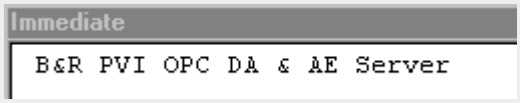
```
Private Sub cmdConnectOPC_Click()
Dim varOPCGroups As OPCGroups     ' variable of type OPCGroups
Dim varOPCItems As OPCItems       ' variable of type OPCItems
Dim setItemID As String

    Set varOPCServer = New OPCServer ' create object "OPCServer"
    varOPCServer.Connect "B&R.PviOPC"  ' connect to OPC server
    Debug.Print varOPCServer.VendorInfo ' display OPC server name
```

Task:

Starting the program for the first time to check the connection to the OPC Server.

After pressing the <**Connect**> button, the server name is displayed in the Visual Basic output window using the line "**Debug.Print varOPCServer.VendorInfo**".



Once the info has been successfully output, the program should be ended and the program code for the "**Connect**" should be added:

```
Debug.Print varOPCServer.VendorInfo ' display OPC server name
' connect varOPCGroups variable to OPCGroups object
Set varOPCGroups = varOPCServer.OPCGroups
' add to collection object a Group(OPCSample)
Set varOPCGroup = varOPCGroups.Add("OPCSample")
varOPCGroup.IsSubscribed = True
varOPCGroup.IsActive = True       ' activate group
varOPCGroup.UpdateRate = 50      ' define refresh with 500ms
```

In the next step, the OPCItem Automation objects are added (i.e. access to the controller's variables).

The folder name and the name of the item from the OPC database should be used as variable name.

```
'connect varOPCItems variable to OPCItems of OPCGroups object
Set varOPCItems = varOPCGroup.OPCItems
If varOPCItems Is Nothing Then    ' exit if no OPCItems available
    Exit Sub
End If
' define item with "Folder.Name" from OPC database
setItemID = "PLC Variables.Visu_Lifesign"
' connect variable to OPCItem(1)
Set varOPCItem(1) = varOPCItems.AddItem(setItemID, 1)
' define item with "Folder.Name" from OPC database
setItemID = "PLC Variables.Visu_PV1"
' connect variable to OPCItem(2)
Set varOPCItem(2) = varOPCItems.AddItem(setItemID, 2)
```

To complete the **connect function**, the <**Connect**> button is locked and the <**Disconnect**> button is enabled.

```
    cmdConnectOPC.Enabled = False
    cmdDisconnectOPC.Enabled = True

End Sub
```

The data items can now be written to and value changes are now automatically registered by the server via events.

> **Note:**
>
> As many items as are needed can be added to the "Item Collection", as long as they are also available in the database.

5.2.6 Disconnect function

The references to the collection objects are queried by pressing the <**Disconnect**> button and cleared using the **Remove** method for the **OPCItem** objects and finally for the **OPCGroup** objects. A **Disconnect** function is then called on the **Server** object.

Program code for the disconnect function:

```
Private Sub cmdDisconnectOPC_Click()
Dim varOPCGroups As OPCGroups    ' variable of type OPCGroups
Dim varOPCGroup As OPCGroup      ' variable of type OPCGroup
Dim varOPCItems As OPCItems      ' variable of type OPCItems
Dim varOPCItemLocal As OPCItem   ' variable of type OPCItem

    ' link OPCGroups variable to server object
    Set varOPCGroups = varOPCServer.OPCGroups
    ' loop to remove all groups of type OPCGroups
    For Each varOPCGroup In varOPCGroups
        If Not (varOPCGroup Is Nothing) Then
            varOPCGroup.IsActive = False
            varOPCGroup.IsSubscribed = False
            Set varOPCItems = varOPCGroup.OPCItems
            For Each varOPCItemLocal In varOPCItems
                Dim ServerHandle(1) As Long
                Dim Errors() As Long
                ServerHandle(1) = varOPCItemLocal.ServerHandle
                ' remove item from collection
                varOPCItems.Remove 1&, ServerHandle, Errors
            Next varOPCItemLocal
            ' remove group from collection
            varOPCGroups.Remove varOPCGroup.ServerHandle
        End If
    Next varOPCGroup
```

```
    For i = 1 To 2
        If Not (varOPCItem(i) Is Nothing) Then
            Set varOPCItem(i) = Nothing
        End If
    Next i
    ' release object variables
    If Not (varOPCGroup Is Nothing) Then
        Set varOPCGroup = Nothing
    End If
    ' release connection to server
    If Not (varOPCServer Is Nothing) Then
        varOPCServer.Disconnect
        Set varOPCServer = Nothing
    End If
    cmdConnectOPC.Enabled = True
    cmdDisconnectOPC.Enabled = False
End Sub
```

When the **disconnect** method is called, the last interface to the OPC Server is enabled and the OPC Server is stopped.

5.2.7 Event function for value changes from the OPC Server

To maintain **value changes** to data items in the OPC Server, the **DataChange event** from the OPCGroup which was set up in the variable declaration with "**WithEvents**" is called **automatically**.

Select the **varOPCGroup** object and the **DataChange** procedure from the program code.

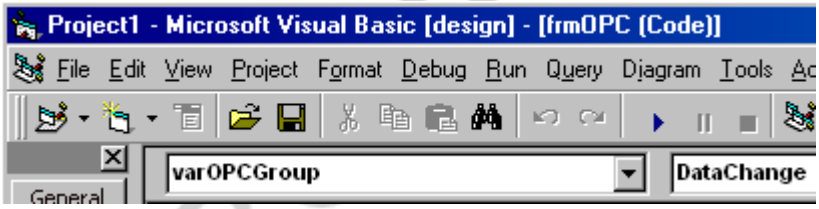


Fig. 40: Data Change event

The following program code should be created for this event:

```
Private Sub varOPCGroup_DataChange(ByVal TransactionID As Long, _
        ByVal NumItems As Long, ClientHandles() As Long, _
        ItemValues() As Variant, Qualities() As Long, _
        TimeStamps() As Date)

Dim varOPCItems As OPCItems
Dim Item As OPCItem
Dim i As Integer
    ' object OPCItems is linked to existing items
    Set varOPCItems = varOPCGroup.OPCItems
    For i = 1 To NumItems
        If (ClientHandles(i) = 1) Then
            'TODO: check the quality here
            txtVisuLifesign.Text = ItemValues(i)
        ElseIf (ClientHandles(i) = 2) Then
            'TODO: check the quality here
            txtVisuPV1.Text = ItemValues(i)
        End If
    Next i
End Sub
```

The values of the data items **Visu_Lifesign** and **Visu_PV1** will now be displayed in the TextBox controls when the program is started and the **connect function** is executed.

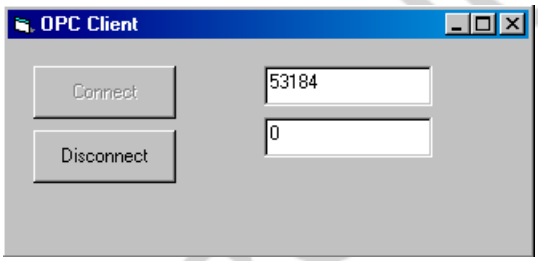


Fig. 41: Displaying values

Remember:

The **ClientHandle** was defined by the **Connect** function when assigning the varOPCItem(**1**).

setItemID = "PLC Variables.Visu_Lifesign"
Set varOPCItem(1) = varOPCItems.AddItem(setItemID, **1**)

5.2.8 Writing to data items

Values are written to a data item using the **Visu_PV1** variable because this variable is not changed in the controller program.

The **Write** method is used for writing to a data item.

Task: Writing to the "Visu_PV1" data item and testing the function.

The **change event** is called by double-clicking on the "**txtVisuPV1**" TextBox.
This event should be changed to "**KeyPress**".

The following program code should be entered:

```
Private Sub txtVisuPV1_KeyPress(KeyAscii As Integer)
Dim WriteValue As Long

    If KeyAscii = vbKeyReturn Then
        WriteValue = Val(txtVisuPV1.Text)
        If Not varOPCItem(2) Is Nothing Then
            varOPCItem(2).Write WriteValue
        End If
    End If
End Sub
```

Caution:

The entry will not be automatically checked. However, the user can do this on his own if necessary.

A numerical value should be entered in the second TextBox **txtVisuPV1** once the program has been started. The value is transferred to the data item by pressing the <**Enter**> key and is therefore also written to the **PV1** controller variable.

Resultat:

Es wird der Wert im TextBox Control „**txtVisuPV1**“ beim Drücken der <**ENTER**> Taste auf die Variable geschrieben.

5.3 Summary of OPC programming

With just a few program lines and a small Visual Basic project, a connection to the OPC Server can be set up and values can be read/written.

The functions and methods that were used in this training module represent only a small portion of the possibilities available with OPC programming, but you can see that reading and writing variables is not hard to do.

More detailed information about programming is available through the OPC Foundation. Help for the OPC Automation interface is not included in the PVI installation.

6. SUMMARY

The B&R PVI OPC Server provides the user with extensive access to the world of B&R control systems.

The possibility for the user to freely select finished **SCADA** visualizations for Windows means that there are essentially no limitations for accessing a controller's variables.

If the possibilities offered in a visualization are not sufficient, then the user still has the option to program the OPC Server (i.e. access to functions, methods and events via the Automation interface).

As a result, it should be possible to meet the demands of the OPC visualization for any application.

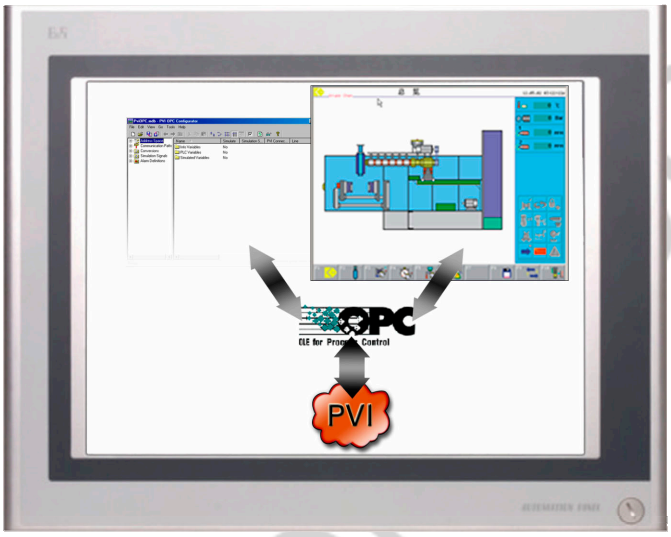


Fig. 42: PVI OPC

7. APPENDIX

OPC
OLE for **P**rocess **C**ontrol

OLE
is the abbreviation for **O**bject **L**inking and **E**mbedding. The term is used for the dynamic linking of objects in different Office applications.

DCOM
stands for Distributed Component Object Model and describes an object model for implementing distributed applications according to the client – server principle. This allows a client to simultaneously use multiple servers and a server to simultaneously provide its functionalities to multiple clients.

OPC database
The OPC configuration is managed in an "access database". "Access" stores all of the database data in a single file with its own .mdb file format.

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 *

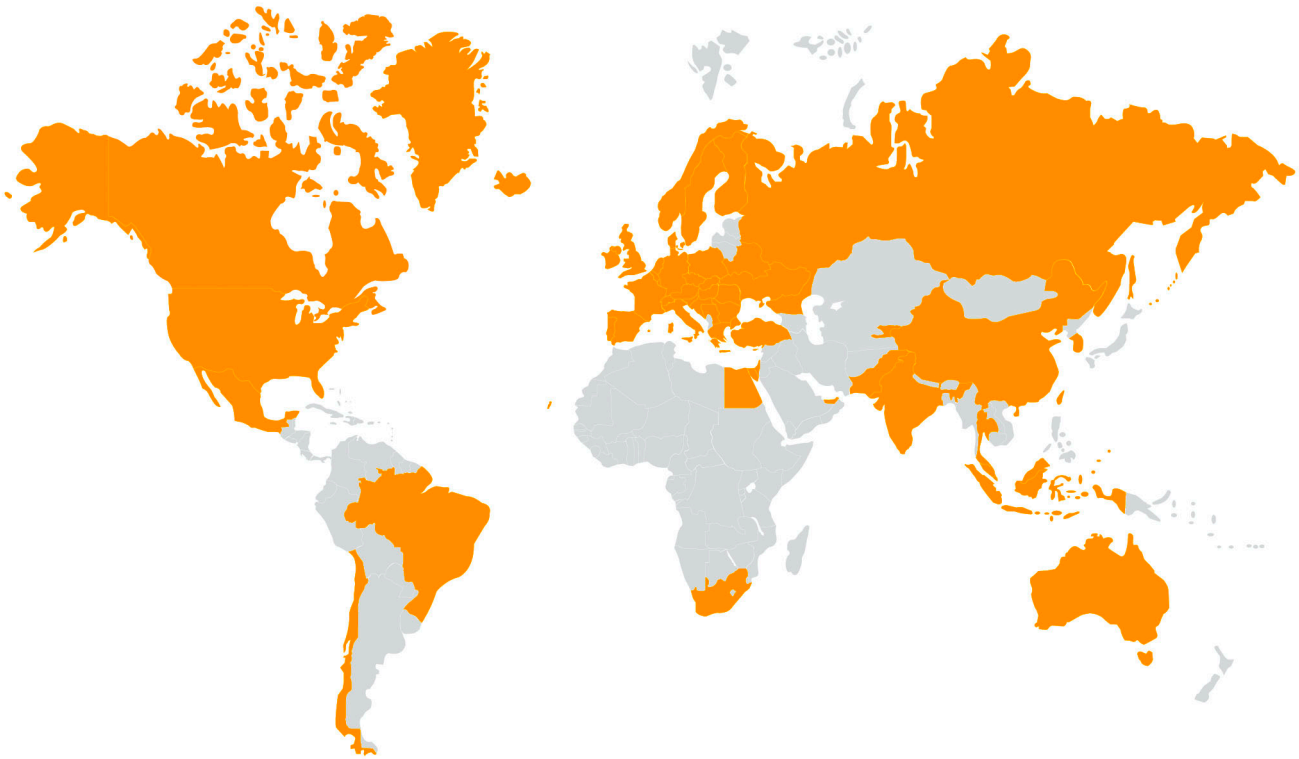
*) upon request
**) see Product Catalog

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

TM730TRE.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