PVI Communication

TM710

Perfection in Automation

www.br-automation.com

Requirements

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

Table of contents

1. INTRODUCTION

To set up and program a visualization so that it meets the necessary requirements, it is important to know the different types of communication between the actual end product "**visualization**" and the **data acquisition** between PVI ⇔ controller.

PVI is more than a communication driver that exchanges data between a PVI client and a controller using one specific protocol.

This training module will illustrate the possibilities and advantages of PVI communication:

- Communication between PVI client and PVI
- Communication between controller and PVI
- Communication between multiple PVI instances (client / server mode)

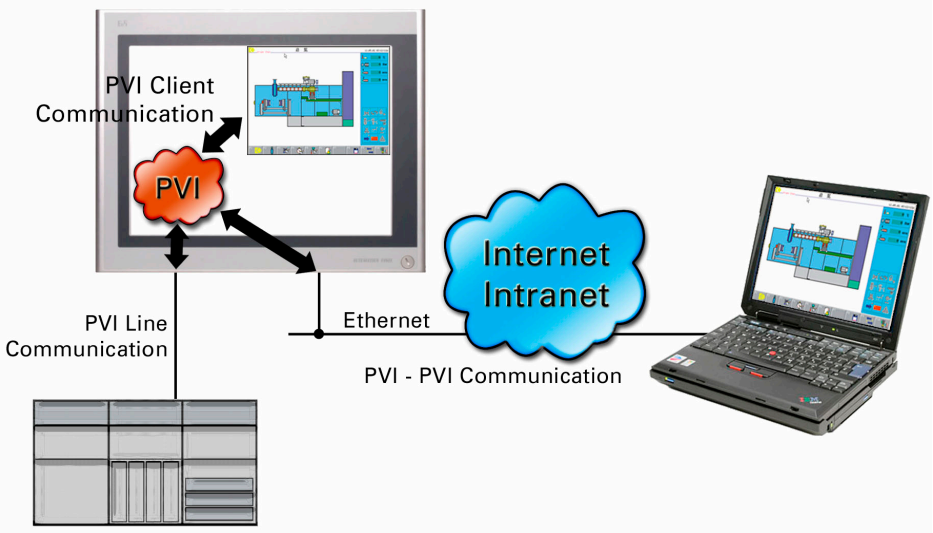


Fig. 1: PVI communication

Brief exercises and tasks will be used to illustrate the possibilities and areas of use for the individual types of communication.

Programming knowledge is not necessary for these exercises. A few of the programs included in the PVI Server&Runtime / Development installation are used as well as Automation Studio.

1.1 Objective

Participants will learn about the different types of PVI communication.

Participants will understand the internal processes of data acquisition, data transmission and data management in order to properly set up and program communication to ensure optimum data transfer speed for their own applications.

After working through the individual exercises, participants will be able to handle advanced programming of the PVICOM interface, PVIServices, PVIControls.NET as well as the configuration and programming of the PVI OPC server.

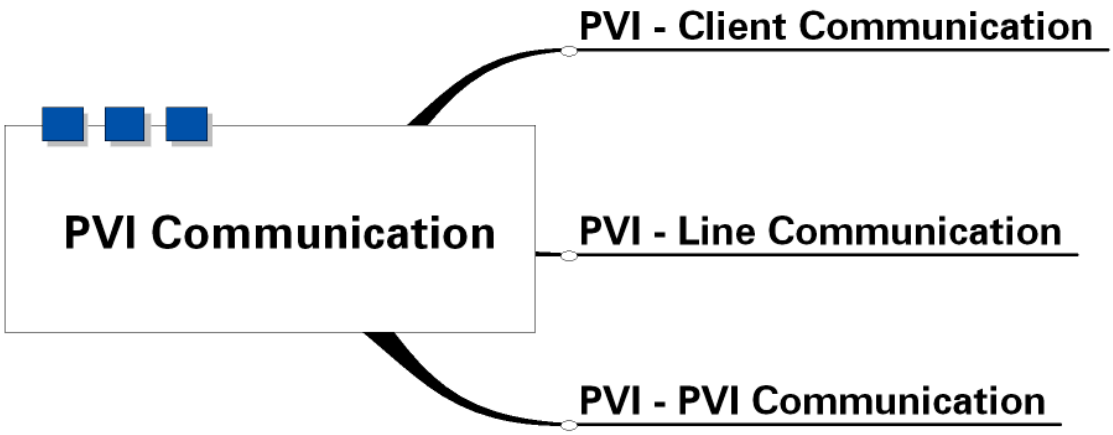


Fig. 2: Overview

With frequent references to the "**PVI help**", participants will also learn how to work with this help structure and how to find detailed information pertaining to their own questions and requirements.

2. PVI – CLIENT COMMUNICATION

Anyone who has programmed a B&R controller has already worked with a PVI client application.

"Automation Studio"

Even Automation Studio is a PVI client in the classical sense, which exchanges data with the B&R controller via a selected medium (serial, CAN, TCPIP, etc).

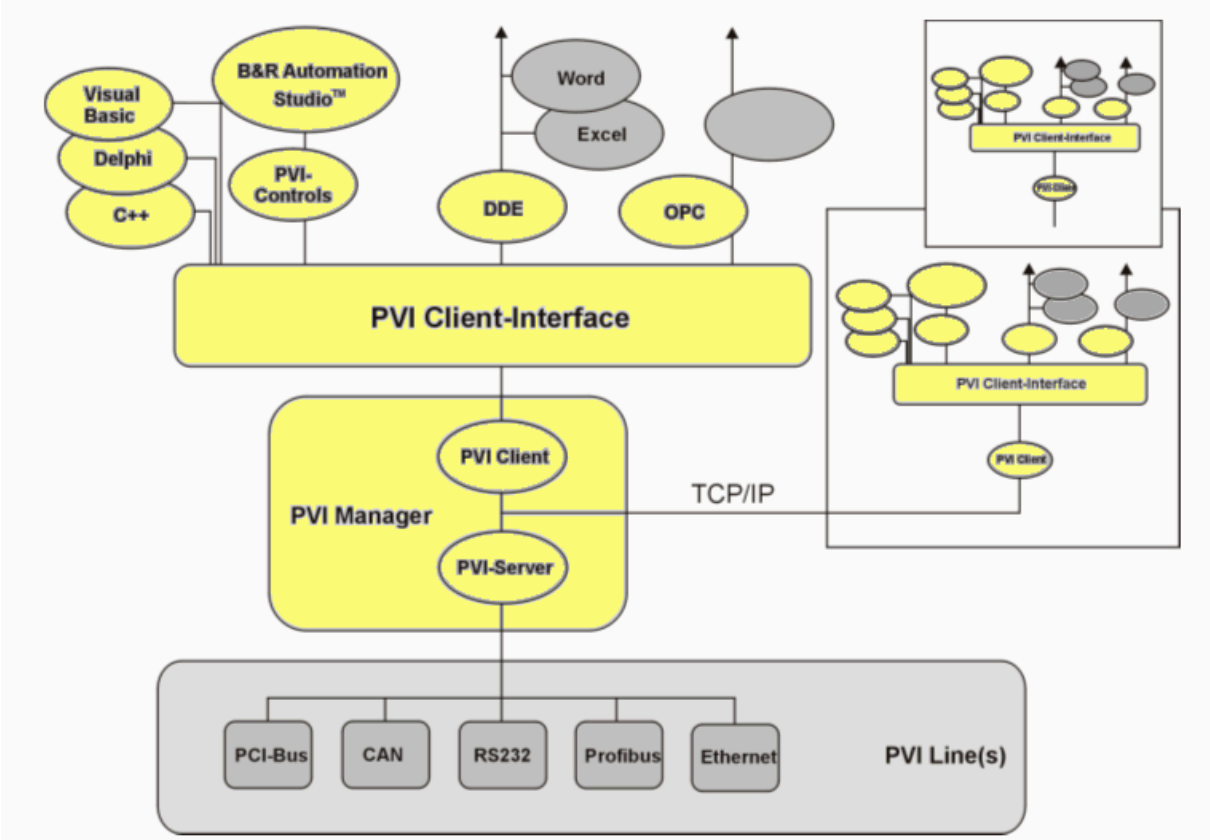


Fig. 3: PVI client communication

This chapter describes the communication between the **PVI client** and the **PVI manager** via the PVICOM interface.

2.1 The PVI Manager

The **PVI Manager** (central component of PVI) handles the management of all types of process data, from simple process variables to lists, programs or data objects.

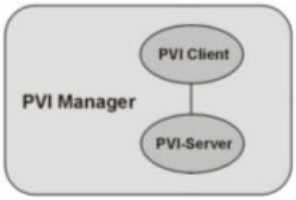


Fig. 4: PVI Manager

The PVI Manager **organizes the process data according to chronology as well as direction**. That means the PVI Manager coordinates the data transfers from the user configuration (direction, protocol, medium, device, etc).

Special attention is given to asynchronous management in order to be able to fit in e.g. network delays for other tasks or coordination of event processing between other tasks.

The **PVICOM interface** (client interface) establishes **access to the PVI** at the lowest level.

Fig. 5: PVI client interface

As a result, this is the most optimum PVI interface in regard to performance.

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

The PVI Manager can be started either as a normal **Windows process** or as a **Windows service**. The only difference is that the PVI Manager as service is not automatically terminated when changing a user in Windows (Log Off).

This option can be selected while installing a PVI package. When the PVI Manager is started, this option is indicated using a different PVI Manager icon in the SysTray.

PVI Manager as process...

PVI Manager as service...

2.2 The PVI object hierarchy

The PVI Manager manages all processes in an object structure (i.e. the **object hierarchy**).

Each **process object** in the hierarchy takes on a specific task and is defined using a unique name (**path name**) and corresponding parameter.

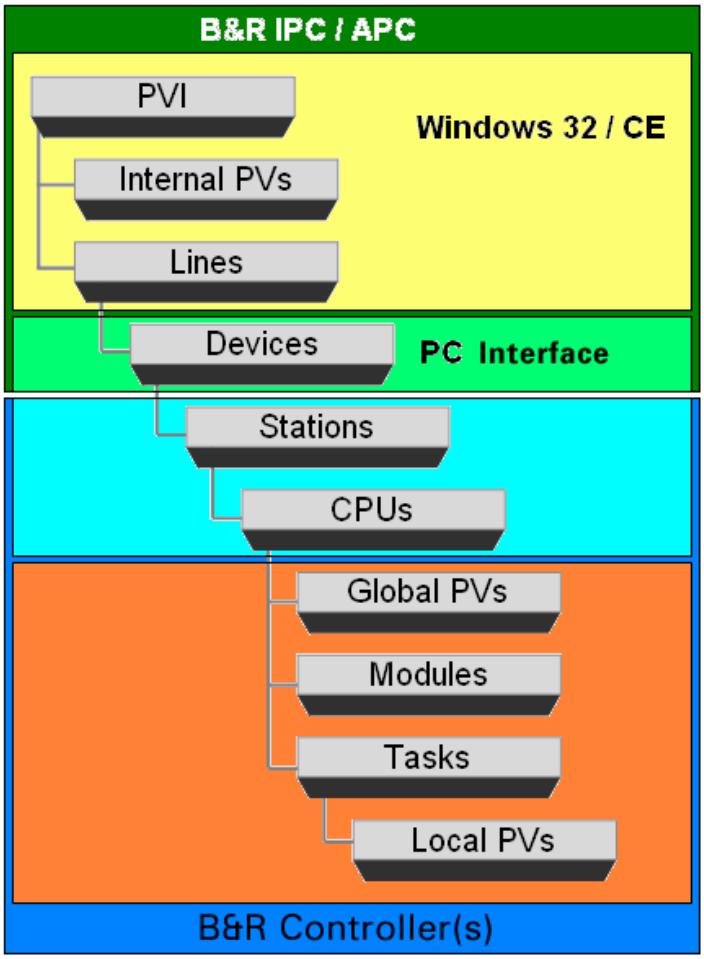


Fig. 6: PVI object hierarchy

Note:

An understanding of the object hierarchy is necessary when programming and setting up a PVI Client application.

2.2.1 Process object

A process object represents either a specific logical/physical part of the communication connection or an object on the controller.

Fig. 7: Process object type - Device

A process object is defined by the following components:

- Object name
- Object type
- Connection description (**CD**)

Note:

The PVI object hierarchy and the definitions for all types of access are identical whether the user employs programming techniques with PVICOM functions via the PVICOM interface or uses a configuration file or configurators to establish PVI server communication.

PviServices makes up the object hierarchy on the user's side in a service, CPU, task and variable class, and is configured by defining the properties.

Exercise:

Learning to understand process objects

The PVIDEMO is used to explain the object hierarchy, object names, object type and the connection description.

The **PVIDEMO.EXE** is located in the following directory:

C:\BrAutomation\Samples\Pvi\Vc\PviDemo\Release\PviDemo.exe

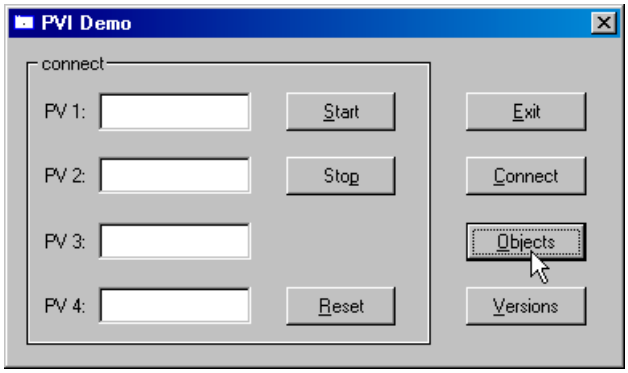
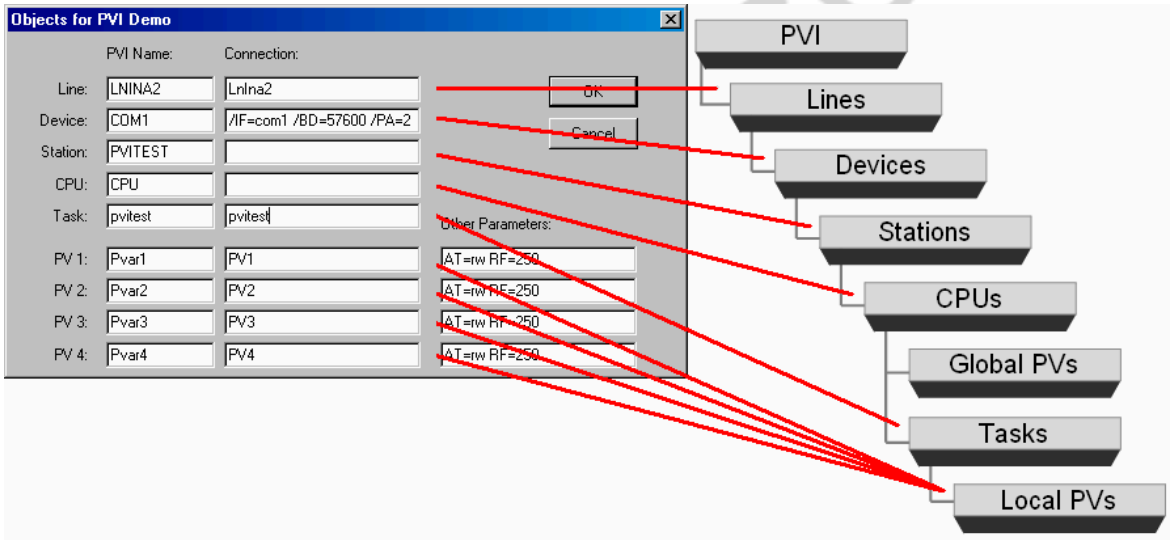


Fig. 8: PviDemo.exe

The configuration window is opened after starting the demo with the "**Objects**" button.



Each line in this window represents a process object.

The **PVI Basis object** does not have to be defined explicity. It is automatically managed by the PVI Manager and is present throughout the entire runtime of the PVI Manager.

The communication protocol is specified via the **Line object**.

Name: @/Pvi/LNINA2
Connection description: CD=LnIna2 (name of the line DLL)

The **Device object** defines the physical connection of the PCs to the outside world.

Name: @/Pvi/LNINA2/COM1
Connection description: CD="/IF=com1 /BD=57000 /PA=2"

The **Station object** represents a station within a network.

Name: @/Pvi/LNINA2/COM/PVITEST
Connection description: CD=""

One or more **CPU objects** define the connection to one or more controllers.

Name: @/Pvi/LNINA2/COM1/PVITEST/CPU
Connection description: CD=""

Variables are mapped on a **task object**, in which the local or global control variables are used in the user task.

Name: @/ Pvi/LNINA2/COM1/PVITEST/CPU/pvitest
Connection description: CD="pvitest" (task name on the controller)

The **Variable object** represents a control variable with any data type.

Name: @/ Pvi/LNINA2/COM1/PVITEST/CPU/pvitest/Pvar1
Connection description: CD="PV1" (variable in the "pvitest" task)

Caution:

Global variables can also be registered directly on the CPU object. However, we recommend setting up all variables in one task object because this considerably speeds up the time needed for registration (identification of the symbolic variable names).

Exercise: Checking the object names and the connection description in the PVI Monitor / Snapshot Viewer:

The SnapShot Viewer is used to diagnose the operating states of the individual PVI objects. More detailed information is available in the PVI User Documentation.

The PVI Monitor is started by double clicking on the PVI Manager icon in the Systray or by selecting PVI Monitor from the shortcut menu.

Steps for this exercise:

- Open the SnapShot Viewer window with the key combination <**ALT**> + <**F9**>.
- Take a snapshot of all of the process objects registered on the PVI Manager as well as their operating state by pressing the <**F5**> key.
- Select the entry "**Process Objects**"
- Show the object hierarchy, the object names and the connection description by expanding the individual objects.

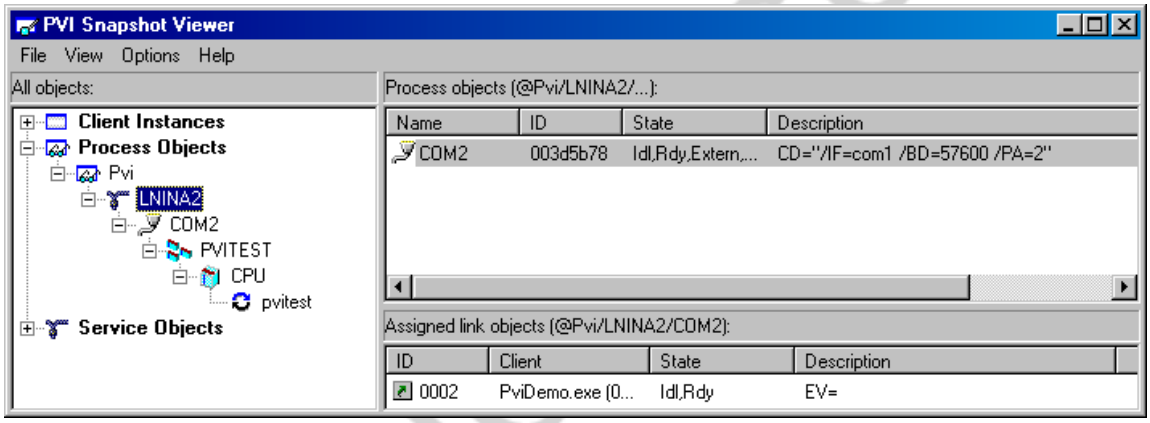


Fig. 9: PVI SnapShot Viewer

Exercise: Changing the connection description

The goal of this task is to change the connection description in the "**PviDemo**" (i.e. the "Connection" column) so that a TCPIP connection can be set up.

The changes can be checked in the SnapShot Viewer.

You can read about the line-specific parameters in the PVI user documentation.

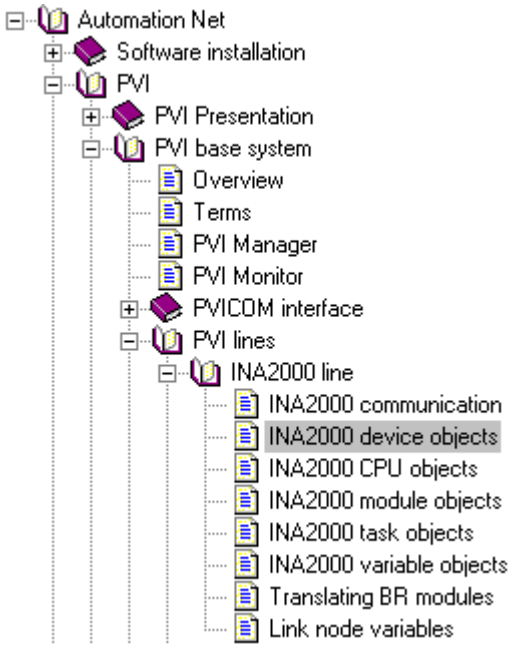


Fig. 10: PVI line documentation

Objective:

- Changing the connection description does not require changing the object names
- Using the PVI Line documentation
- Using the PVI SnapShot Viewer to check and to evaluate the PVI objects

Result:

- The object name is a symbolic name and is therefore independent from the connection description
- Device object:
  Name: @/Pvi/LNINA2/COM1
  Connection description: CD="/IF=TCPIP /SA=1"
- CPU object:
  Name: @/Pvi/LNINA2/COM1/PVITEST/CPU
  Connection description: CD="/DA=2 /DAIP=10.0.0.2"

2.2.2 Static and temporary process objects

A process object can be created as a **temporary** or **static** process object. The method is the same for both types.

Static process objects

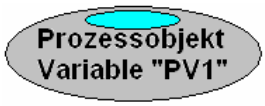
A static process object is only set up once and remains throughout the entire runtime of PVI Manager.

Static process objects are advantageous in the following situations:

- A PVICOM application which is started and stopped many times and which needs a large number of process objects
- The object structure should be managed centrally. At the start (e.g. Windows start-up), a generator or configurator (as the PVICOM application) sets up all required process objects. Then the application can be ended again.
- An interface server which remains active during the entire PVI Manager runtime and provides information about the necessary object structure right after being started.

Temporary process objects

A temporary process object is set up together with a link object (= active process object). When releasing the link object or when terminating the PVICOM application, the temporary process object is also deleted.



Temporary process objects are advantageous in the following situations:

- A PVICOM application which is not started and stopped many times or which needs just a few process objects
- Process objects which are only used briefly in a PVICOM application and which are seldom needed (e.g. for carrying out services)
- If the connection description of a process object is modified during runtime, then it and all subordinate objects should be set up as temporary process objects.

The points mentioned above are only intended as decision guidance. The type of process object used by the PVICOM application needs to be decided according to requirements. However, if in doubt, temporary process objects are preferred since they are more easily handled.

Note:

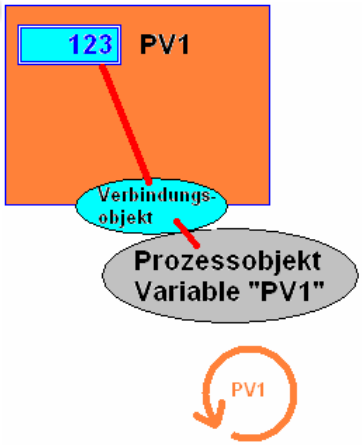
The **PVI OPC Server** and the **PVIControls** only use static process objects. The user can define this by setting a property in the **ACConfigurator** of the **PVIControls.NET**.

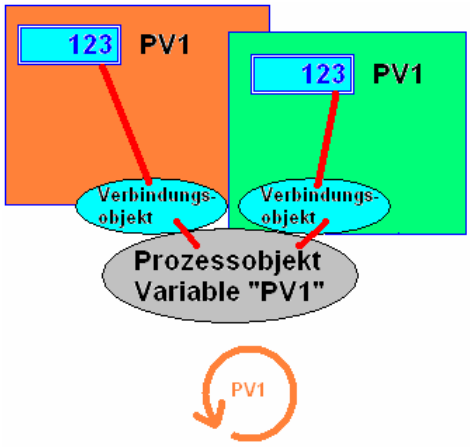
Example:

A visualization with multiple screen pages is a classic example of using static and temporary process objects.

Using static process objects:

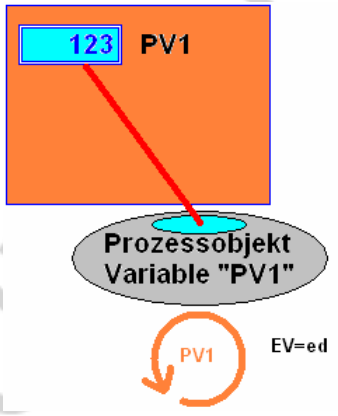
- When starting the visualization, all objects are set up as static (i.e. only the process object is created – no link object).
- To display a variable on a screen page, a link object must now be created on the corresponding variable process object.
- The variable will now be read and monitored by the PVI Manager / PVI Line. A data change is received in the response data of the link object and can be evaluated accordingly = **active process object**.
- When changing pictures, the link object is cleared again, but the process object remains. Variable reading is deactivated = **inactive process object**.
- Another link object is created on the same process object if it is necessary for the same variable to be displayed multiple times. This makes it possible to display and close the screen pages independently from one another.
- Another advantage of this method is that each link object can have a different scaling or data type.

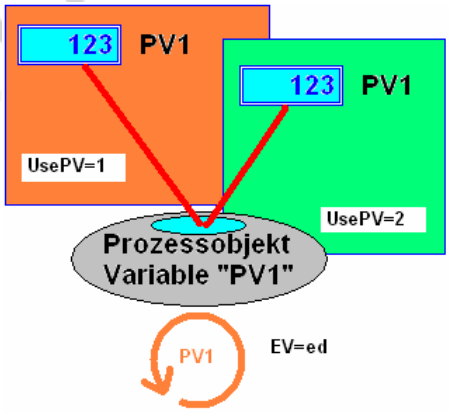




Using temporary process objects

- All process objects are set up temporarily when starting (i.e. the process object already has a link object).
- If a variable is displayed on a screen page, then the process object is switched to "**active**".
- The variable will now be read and monitored by the PVI Manager / PVI Line. A data change is received in the response data of the link object and can be evaluated accordingly = **active process object**.
- When changing a picture, the process object is switched to "**inactive**" again (i.e. variable reading is deactivated = **inactive process object**).
- If the same variable must be displayed multiple times, then the client application must be used to monitor whether the variable is displayed multiple times. The process object can be switched to "**inactive**" when the variable is no longer required.





Note:

Exercises for static and temporary process objects are performed in the training modules TM711 (PVICOM Programming) and TM712 (PVIServices).

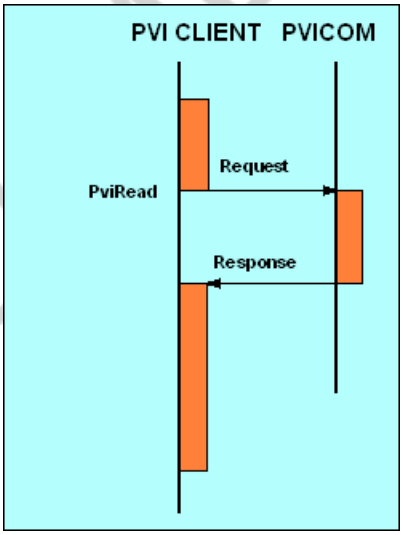
2.3 Synchronous / asynchronous data acquisition

All tasks are processed "**asynchronously**" within the PVI Manager and the line. This means that **delays** do not occur and other tasks can be shifted.

However, it can sometimes be necessary to execute a task "**synchronously**" in the PVI Client application (e.g. read or write request from variables).

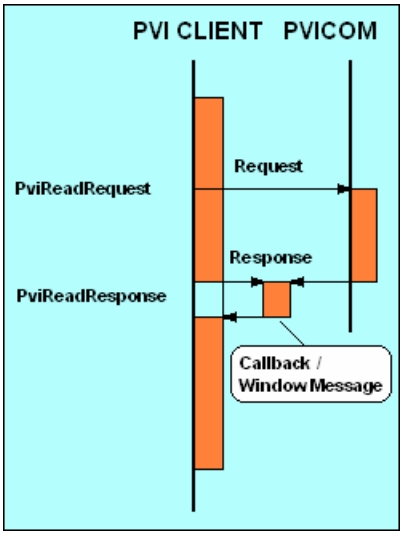
Synchronous tasks:

- Synchronous tasks wait for the task's confirmation (response data) in the function call.
- No other system operations are possible during this time.
- Synchronous function calls in a loop should be avoided.
- Synchronous tasks cannot be "blocked" in the PVI line (i.e. each read or write access is sent to the controller in a separate communication frame).



Asynchronous tasks:

- In asynchronous tasks, the application program is further processed after the function call.
- The confirmation is made "asynchronously" to the program execution in a separate function that is called automatically. The data from the read tasks can be read in this function or a confirmation from a write task is provided.
- Asynchronous tasks can also be read in a loop.
- Asynchronous tasks can be blocked from the PVI line.



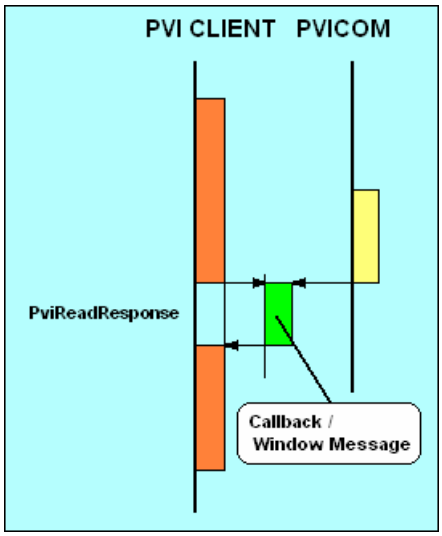
2.4 Notifying the client of a data change

The PVI Client application is notified when event data is present and when data is changed in "**active**" process objects.

Different user messages are available depending on the programming environment being used:

User messages	Description
Windows message	Signals using a Windows message. Response or event data is read using the corresponding response function. This is the preferred method for Visual C/C++ applications.
Callback with data	Signals using a callback function. Response or event data is transferred using the callback function. This is the preferred method for Visual Basic 6.0 applications.
Callback without data	Signals using a callback function. Response or event data is read using the corresponding response function.
Asynchronous callback	Signals using an asynchronous callback function. Response or event data is read using the response function. This method should be used for Visual C/C++ applications without windows.

A PVICOM application can also use the different user message variants with one another. For example, a Visual C/C++ application can receive event data as a Windows message and response data as callback.



2.5 Communication performance

When setting up a PVI Client application, suitable configuration of each individual variable can strongly influence the communication performance.

> **Note:**
>
> Exercise examples about the possibilities will be provided in the corresponding training modules for PVICOM Programming TM711 and for PVIServices TM712.

2.5.1 Active / passive switching of variables

The PVI Manager or the PVI Line only monitors the process objects / variable objects that are switched to active.

The fewer variables that are active, the faster the active variables can be refreshed.

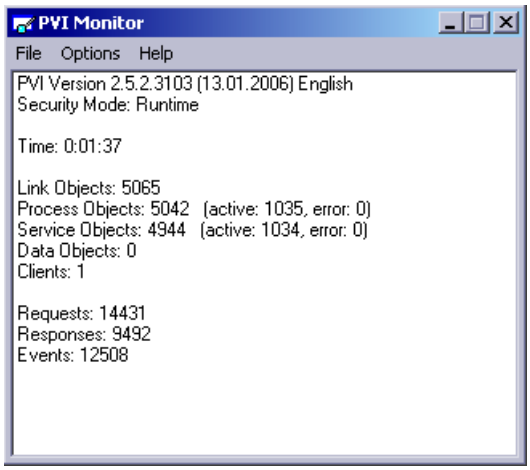
Which variables should be switched to active:

- Variables that are displayed on a screen page
- Variables that must be read in the background of the application (e.g. alarm and trend data)

Which variables should be switched to inactive:

- Variables that are only requested by the controller or written to the controller when specific actions occur
- Variables that are not displayed

The PVI Monitor can be used to evaluate the number of process objects registered to the PVI Manager and the number of active process objects.



2.5.2 Refresh time for variables

The refresh time determines **how frequently a variable object is updated**.

The PVI Manager / PVI Line attempts to acquire the process data within this time period. The controller monitors the event variables (see 3.1) for data changes within this time period.

Caution:

The defined refresh time is not a guarantee of the actual acquisition time because this can be affected by other factors:

- Controller response time
- Number of active variables
- Tasks, asynchronous to the normal read cycle of variables
- The protocol/medium being used

Refresh time	Description
-1	The data is not read automatically. However, the application can control data acquisition using targeted read tasks.
0	Process data is only read once. Reading takes place after the line has initialized the variable object. As with RF = -1, the application can control further data acquisitioning using targeted read tasks.
> 0	The variable object is cyclically provided with updated process data in the specified refresh time (given in ms). The refresh time should be set to correspond with the requirements of the process data (how current).

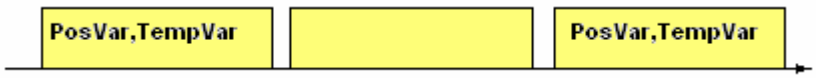
Setting refresh times

- Data that changes slowly (e.g. temperature values) should have a corresponding refresh time in the seconds range.
- Variables that only have to be read when starting the application, such as set values, should be registered with a refresh time of "= 0".
- Data that only has to be read after a specific application action should be registered with a refresh time of = "-1".
- Data that is used to display movements should be registered with a short refresh time.

Example:

Reading position data and temperature values.

PosVar Refresh time = 200ms
TempVar Refresh time = 200ms



All variables are read in the same read cycle. Data changes that are needed at a high speed can also be read more often by setting the refresh times accordingly.

PosVar Refresh time = 200ms
TempVar Refresh time = 5000ms

In this image, it is evident that the variable "PosVar" can be read in almost every communication frame, unlike the variable "TempVar" which is only contained in each n^{th} communication frame.

Caution:

Real-time communication between the PC and controller is not possible with PVI because the defined times cannot be guaranteed due to the dynamic setup of the variables and communication between PVI ⇔ Client application.

2.5.3 Object attribute

Object attributes control the approach when acquiring and handling process data of external variable objects.

The communication performance between the PVI Client application and the PVI Manager and between the PVI Manager and the controller can be improved with targeted application of these attributes. The most important attribute for communication performance is the attribute "e" for activating the event-controlled communication (see also 3.1).

Attribute	Description
r	Allows read access to the process data of a variable object. If this attribute is not specified, a read access attempt is rejected with an error. PVI Manager does not carry out any cyclic read tasks.
w	Allows write access to the process data of a variable object. If this attribute is not specified, a write access attempt is rejected with an error.
e	**Operation mode PLC event variable. The PLC monitors the process data for changes. Therefore, the PVI Manager does not need to execute any cyclic read tasks (only INA2000 and NET2000 Line).**
h	Fast echo after write access. The time between a write task and a resulting data change event can be reduced with the "h" object attribute. However, this cannot accelerate the actual write task. The write data is compared with the process image before the transfer to the PLC and, if there is a change, a data change event is triggered via all existing link objects.
d	Direct event trigger for the POBJ_EVENT_DATA event. The "d" attribute is only effective in connection with the "e" attribute. A direct event trigger means that all process data returned by the line or PLC is sent directly to the application. Application: e.g. life sign monitor on CANDirect Line

2.5.4 Variable hysteresis

A filter can be defined for a value change via the event hysteresis. If the event hysteresis is defined in the variable object, a change event from the PVI Manager to the PVI Client application is only triggered if the value change is greater than or equal to this value.

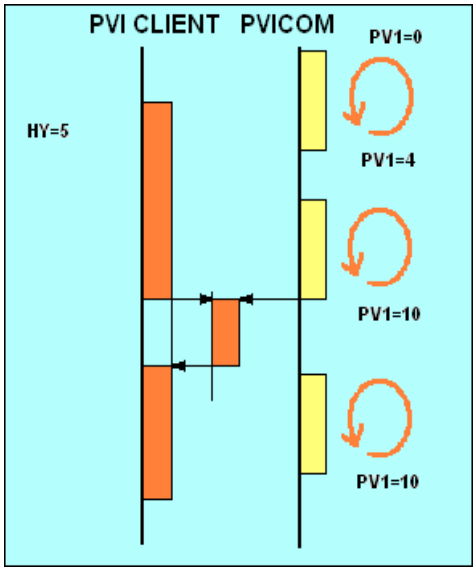


Fig. 11: PVI hysteresis

As can be seen in this image, when the hysteresis has the value 5 (HY=5) only a data change greater than or equal to this value (e.g. from 4 to 10) triggers a message to the PVI Client application.

Note:

If the hysteresis is used with the event-driven communication (attribute = e), then the controller monitors the value change with the defined hysteresis.

2.5.5 Using structures and arrays

A few things must be taken into considersation if control variables with the data type "**Structure**" or "**Array**" are registered from the PVI Client application:

- A certain structure alignment may be specified depending on the programming environment (e.g. Visual Basic 6.0 ⇔ Alignment = 4).
- Structures and arrays cannot be registered as event variables (attribute = e).
- These data types should be registered with a refresh time "0" or "-1" preventing them from being read cyclically.
- If only a few elements of a structure or an array are changed on the controller, then these elements should be registered as individual variables.
- "**Consistent**" transfer of an entire structure or array is not possible (i.e. the image of the PVI Client application and the controller is not necessarily identical - see 2.6).
- "Simultaneously written" access from the controller and the PVI Client application should be avoided.

2.6 Data consistency

When transferring data between the controller and a PVI Client application, the data transfer must be "**consistent**".

Note:

Only one consistency to scalar data types is provided (SINT, INT, DINT, REAL, etc) for PVI communication with the INA2000 or NET2000 Line.

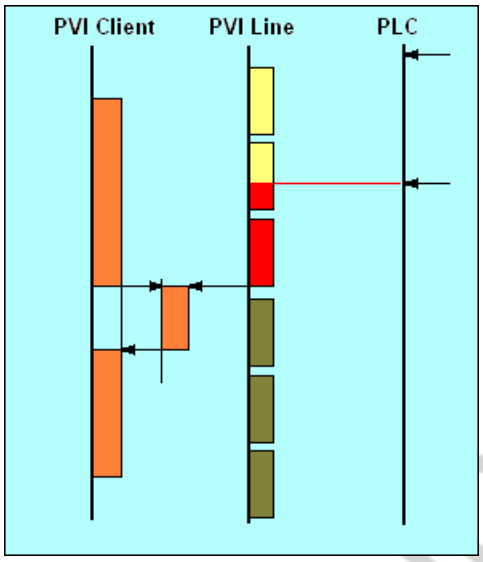


Fig. 12: Data consistency

As you can see in this image, multiple communication frames are required to transfer the structure. If an element in this structure is changed on the controller, then the image of the structure is no longer consistent during the transfer.

In this case, the application must handle the data consistency by synchronizing the data exchange. For example, a separate write or read image could be used to do this, whereby access is made via trigger variables.

Caution:

Unique value changes (incremental value change) should always be evaluated when synchronizing the data exchange. Synchronization based on a BOOL variable can be overseen due to asynchronous access of the PVI Line to the controller.

3. PVI – LINE COMMUNICATION

This section describes the course of communication between the PVI Manager, the PVI Line and the controller from registration of the process objects to polled or event-driven data transfer.

The INA2000 Line is used as basis because this PVI Line is the most commonly used type of communication (online protocol to SG3 and SG4 controllers).

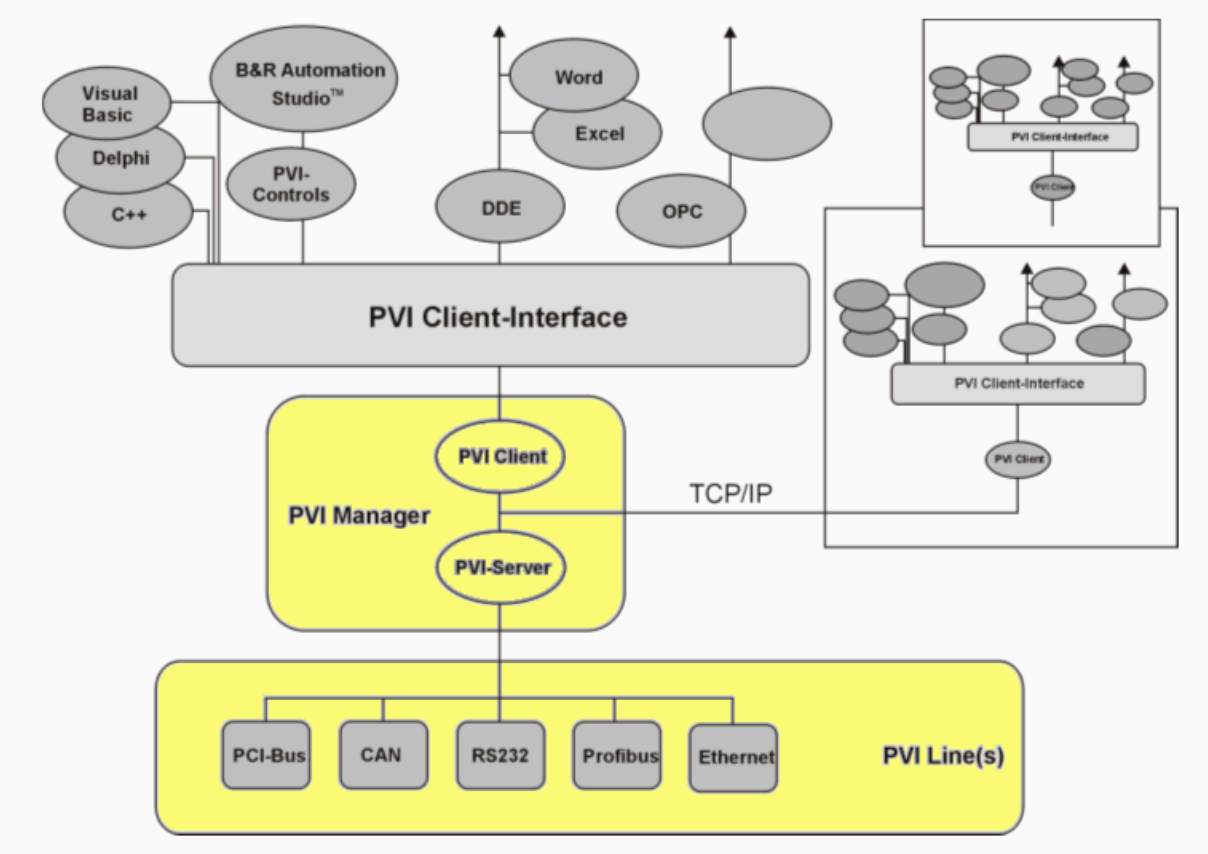


Fig. 13: PVI Line communication

3.1 Establishing connection

Each newly setup process object with line connection triggers a process of **establishing connection** (station and CPU process object) or of **object identification** (module, task, and variable process object).

Depending on the line being used, this process can take some time to complete.

> **Note:**
>
> INA2000 Line: It is recommended to set up all variable objects in one task object. This enables faster identification of variables than if they are set up in the CPU object.

If the process object is enabled and then freshly set up again, then this process has to be repeated. It is therefore an advantage if PVICOM applications only have to set up process objects once.

A disrupted connection (unplugging the cable between the PC and controller) also causes (after the response timeout has expired following x number of the configured or defined repetitions) a new attempt to establish a connection as long as the connection to the controller has been re-established.

If an object is created temporarily, an automatic read task (active process object) is performed after the object has been identified and therefore also a read task to the controller. A data event is sent to the PVI Client application after a data change.

Static objects are only identified. The object is only read and the data event sent to the PVI Client application after a link object has been created.

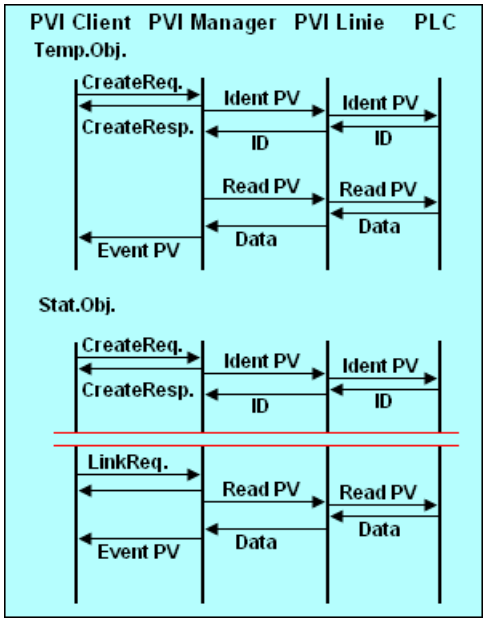


Fig. 14: PVI Verbindungsaufbau

3.2 Polled communication

If a process variable arrives with active event mask and a refresh time > 0, then the PVI Line starts cyclically reading this active process variable from the controller.

As you can see in this image, the process variable "**PV1**" is read cyclically from the controller.

The PVI Manager compares the value of the internal process image with the read value and sends a data event to the PVI Client application anytime a data change occurs.

Multiple active variables are simultaneously read from the controller in a request frame.

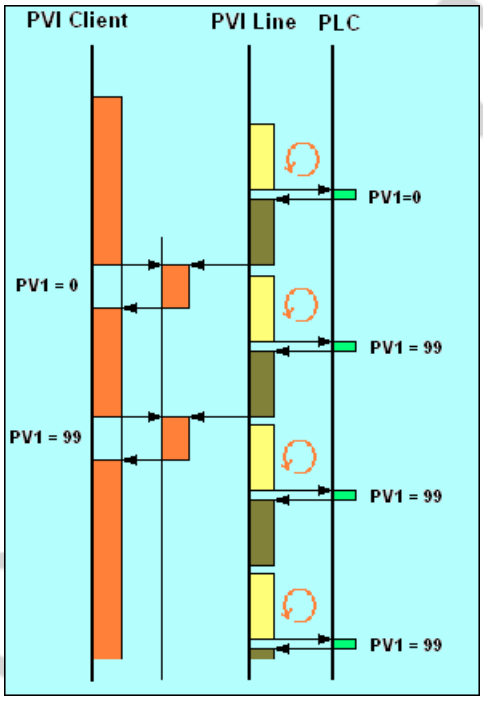


Fig. 15: PVI – Pollende Kommunikation

Advantage of polled communication:

- Quicker update of variables with shorter refresh time

Disadvantage of polled communication:

- When there is a large number of active variables, the defined refresh time for all variables is no longer guaranteed because all variables cannot be read in one request from the controller.

3.3 Event-driven communication

INA2000 and NET2000 Line offer the possibility of event-driven communication (attribute "e").

This view shows the communication between PVI and the controller when only event variables are used.

In this case, only one life sign monitor is active between the PVI and the controller with the Response Timeout (/RT=x) defined for the CPU process object.

The controller monitors every event variable with the defined refresh time (RF=x) in the controller's idle time.

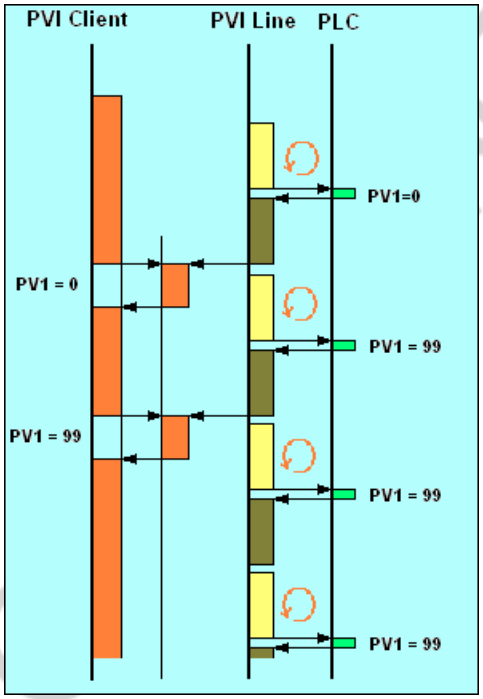


Fig. 16: PVI – Event-driven communication

When a data change occurs on the controller, the PVI is notified in the response frame of the life sign monitor or a "normal" data response that event variables have changed. These variables are then read by the PVI Line.

Advantages of event-driven communication

- Variables that do not change often (e.g. alarm variables) reduce the load on the cyclic communication.
- This allows variables with a short refresh time to be read faster.
- No load on the controller because monitoring takes place in the idle time.

Note:

The different registration of variables as "**polled**" or "**event-driven**" makes it possible to achieve optimum communication performance for all requirements.

4. PVI – PVI COMMUNICATION

PVI offers a **PVI remote connection** for remote system maintenance, for a client / server application between two or more PC's or for programming via network.

The only requirement is a TCPIP connection between the PC's.

This makes it possible to create a connection to multiple PVI server PC's from one PC or to access a PVI server PC from multiple PC's.

In this case, the **PVI Manager** and the **PVI Client application** are located on **different PC's**.

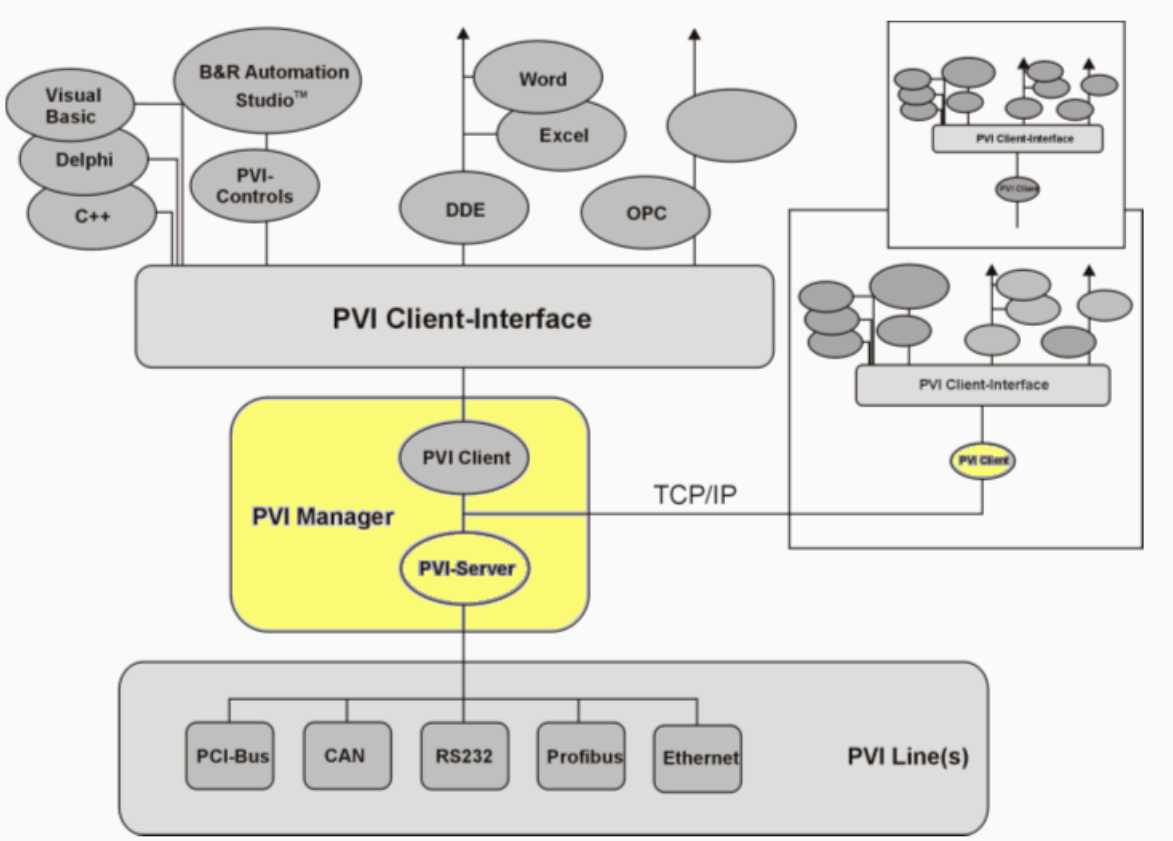


Fig. 17: PVI Client – Server communication

4.1 Topologies for a remote connection

As described in the introduction to this chapter, a PVI Remote connection can be implemented as simple client / server architecture as well as multi-client / multi-server architecture.

The following diagram illustrates simple client / server communication, which is used for e.g. remote maintenance or programming.

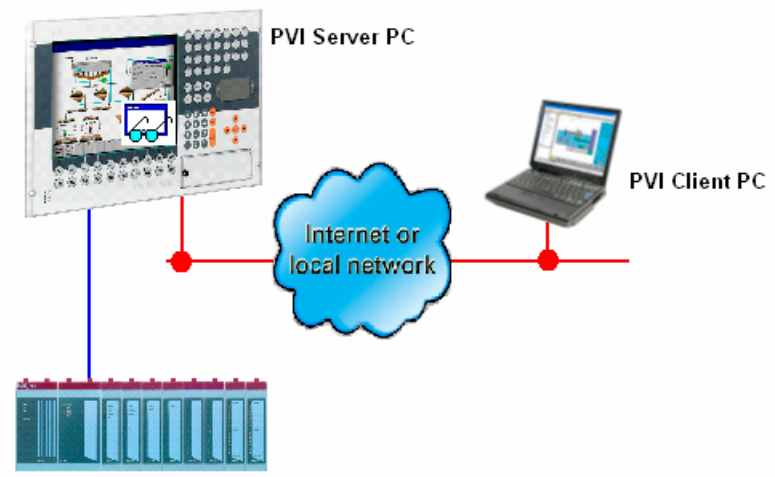


Fig. 18: PVI Remote connection - client / server

Note:

A PVI license (dongle) is not required on the PVI Client PC because the PVI Manager only runs on the PVI Server PC.

This example shows a multi-client / server architecture, whereby a local PVI application is running on one PC but a PVI client application can also run.

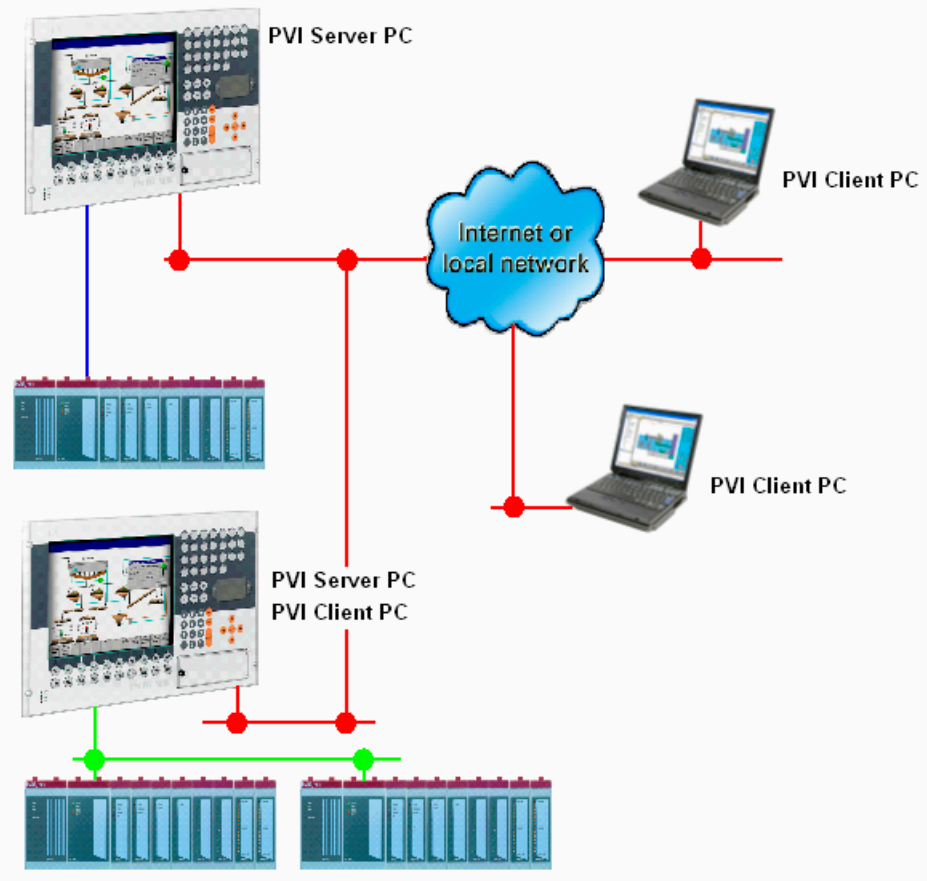


Fig. 19: PVI Remote connection – multi-client / multi-server

4.2 Setting up a remote connection

The only requirement for setting up a remote connection is existing TCP/IP communication between the involved PC's.

It is irrelevant whether this is a direct connection via a network cable or a dial-up connection via a modem.

Exercise: Connecting 2 PC's and testing the TCP/IP connection

Connect 2 PC's via a crossed network cable. Define a separate IP address on each PC.

PC #1:

IP Address: 10.0.0.1
SubNet Mask: 255.255.255.0

PC #2:

IP Address: 10.0.0.2
SubNet Mask: 255.255.255.0

Use the PING command to test the connection (caution: check firewall settings).

<Start> / <Run> : cmd

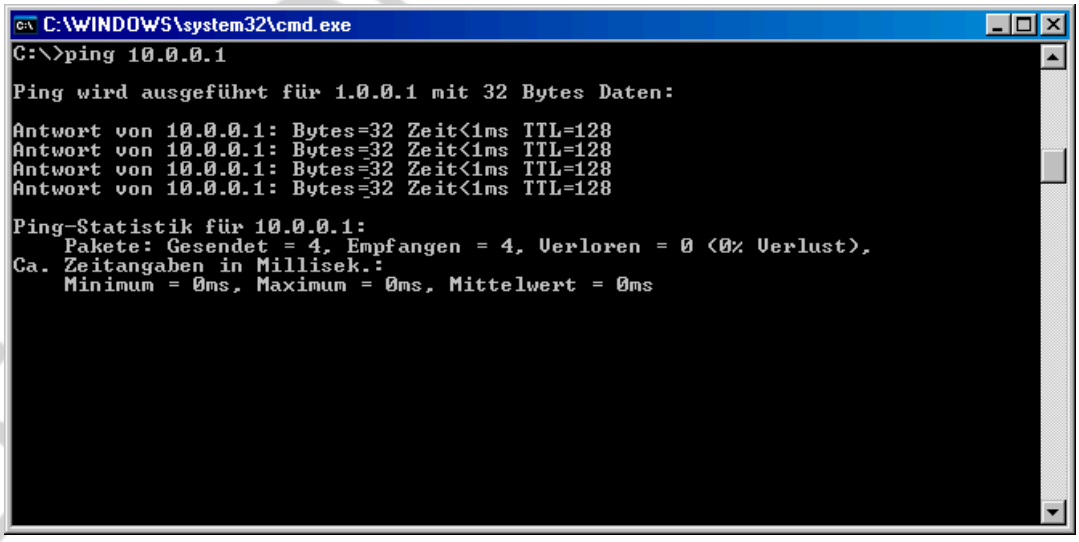


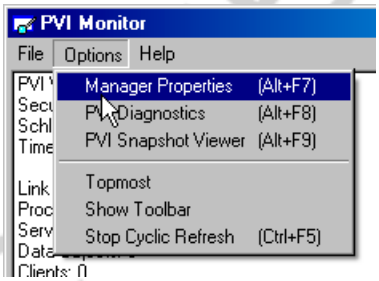
Fig. 20: Windows command prompt: Execute PING command

4.2.1 Configuring a remote connection on a server PC

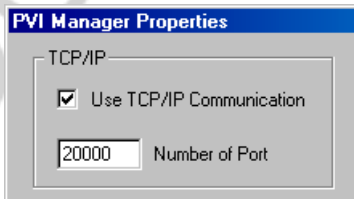
A remote connection on a server PC is set up in the PVI monitor.

The PVI Monitor is started via <**Start**> / <**Programs**> / <**BrAutomation**> / **PviMonitor** or by double-clicking on the PVI Manager icon in the SysTray.

The properties dialog box for the PVI Manager is opened via the menu <**Options**> / <**Manager Properties**> or via the key combination <**ALT**> + <**F7**>.



The **port number** for the remote connection must be defined in this dialog box.



Caution:

Port numbers < 1024 are used by Windows. A port number > 10000 is recommended.

Exercise:

Setting a port number in the Manager properties

Set a port number (default: 20000) in the PVI Monitor for a remote connection.

The PVI Manager must then be terminated and restarted. Any changes to the PVI Manager properties will only take effect after restarting the PVI Manager.

A configured remote connection is displayed after starting the PVI Manager on the server PC.

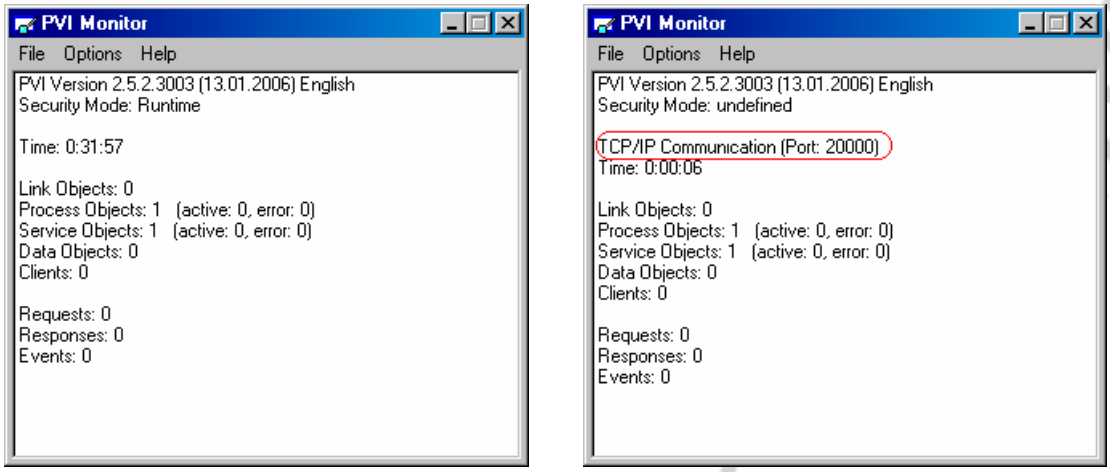


Fig. 21: PVI Monitor – TCPIP connection

From this point on, a PVI Client application can access the controller via a TCP/IP remote connection

Caution:

On a local connection, the PVI Manager is automatically started via the "PviInitialize()" function when accessing the PVICOM interface.
On a remote connection, the PVI Manager must be started manually (e.g. Windows Startup menu or start PVI as a service).

4.2.2 Setting up a remote connection on the client PC

A remote connection is defined in the PVI Client application by also specifying the **IP address** / **computer name** and **port number** of the PVI server PC in the "PviInitialize()" function.

Note:

The definitions are made in the PVI process object's dialog boxes if an installed PVI component (e.g. PVIControls, PVI OPC configurator, etc) is used. Additional programming is not necessary.

Exercise: Testing a remote connection

In the previous exercise, a PC was set up as PVI Server.

The familiar "**PviDemo**" can now be used to test a remote connection.

C:\BrAutomation\Samples\Pvi\Vc\PviDemo\Release\PviDemo.exe

After starting the **PviDemo**, the <**Connect**> button is used to open a dialog box in which a local or remote connection can be set up.

The IP address and the port number of the PVI server PC's are defined.

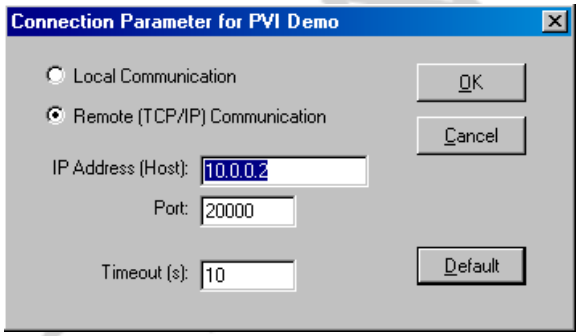


Fig. 22: PviDemo.exe – Remote connection

Result:

- Error 12095 (Communication interrupted. TCP/IP error) is output after the timeout has expired if the PviDemo is started without a network connection.
- When a valid connection is present, the process objects that were set up via the remote connection are displayed on the PVI Server PC in the PVI Monitor.
- The PVI Manager was not started on the PC where the PviDemo was started.

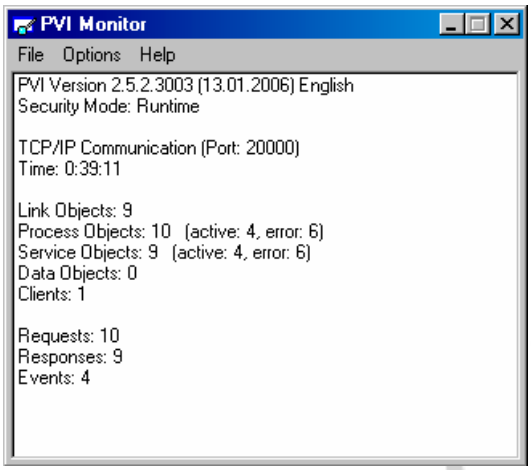


Fig. 23: PVI Server PC – Showing the process objects

Note:

If only one PC is available for this exercise, then a remote connection with only one PC can be tested by entering the IP address "127.0.0.1".

4.2.3 Global events

The global events are used to inform a PVI Client application of the connection status to the PVI Manager and should be used in each PVI Client application which is intended to run as a remote application.

Event	Description
CONNECT	The communication instance (client) established the connection to the PVI Manager (server).
DISCONNECT	The connection between the communication instance (client) to the PVI Manager was interrupted (server). The corresponding error code is reported with the event data. The communication timeout plays the main role in recognizing the disruption in the client/server connection (PviInitialize).
ARRANGE	This event signals that the communication instance has been newly registered with the PVI Manager. This event is triggered if there is a valid client/server connection.

The global events CONNECT and DISCONNECT signal the state of the client/server connection.

The ARRANGE event signals the application when connection objects and temporary process objects have to be set up again.

If there is a longer interruption in the client/server connection (doubled communication timeout), then the PVI Manager automatically de-registers the communication instance and releases all assigned connection objects. If the connection is established later, then the PVICOM application has to set up all objects again. The ARRANGE event can be used to automatically control this procedure.

Note:

More information about the can be found in the PVI User Documentation.

5. SUMMARY

After working through this training module, participants understand the communication between their PVI Client application and the controller.

Either an existing application can be optimized by making specific changes to the individual objects or can be set up optimally from the start when programming a new application.

This training module is also the basis for continued programming of the PVICOM interface (TM711, TM712) and for the use of the PVI OPC server / configurator (TM730).

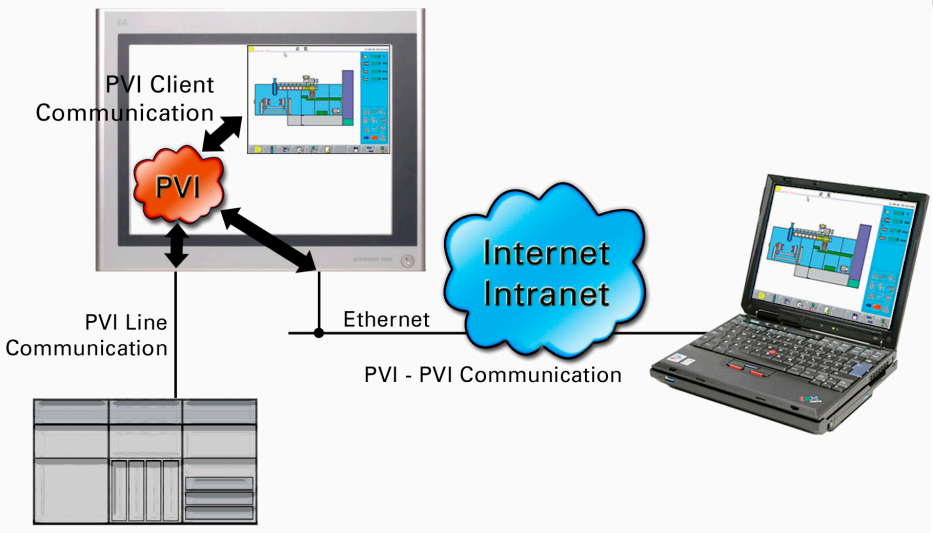


Fig. 24: PVI communication

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 *

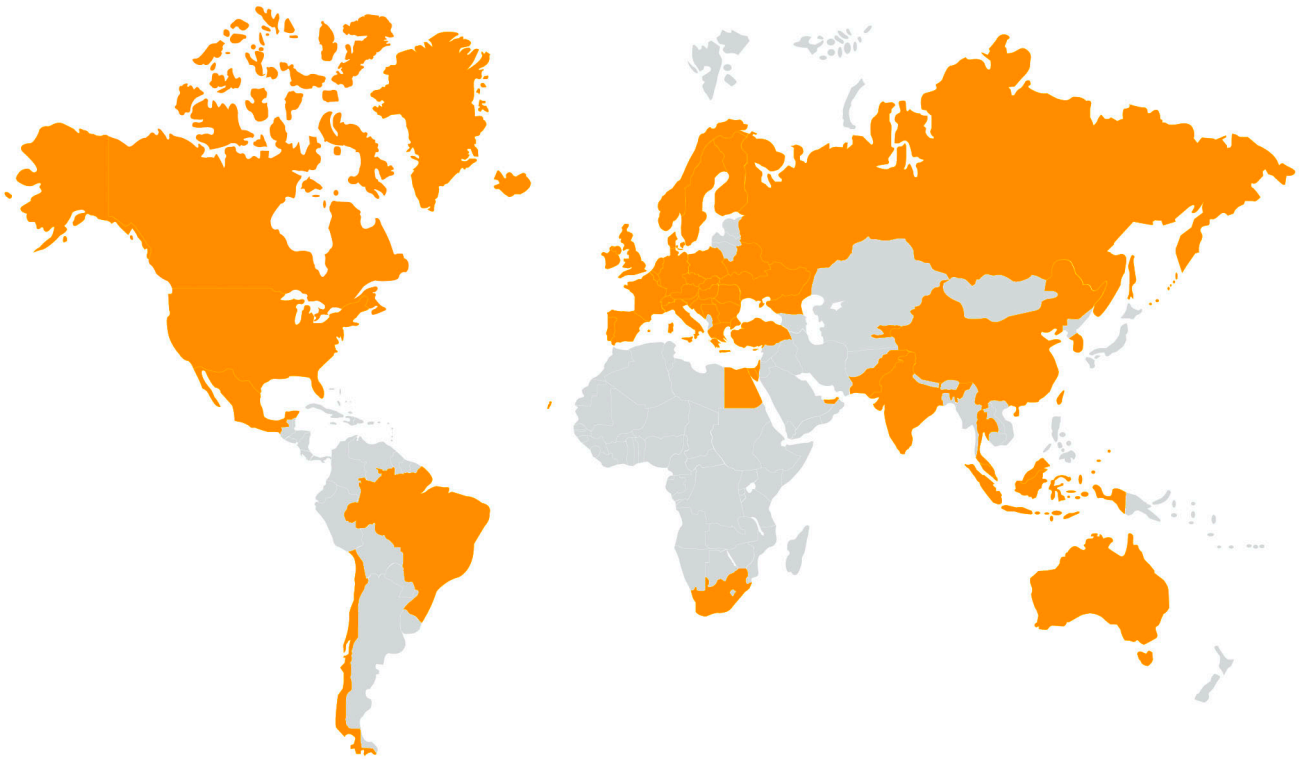
*) upon request
**) see Product Catalog

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

TM710TRE.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