PVSS II
by ETM
www.pvss.com

Quick start

Getting Started - Basics

Installation - First steps for beginners

PVSS II – Process visualization and control system

Version 2.0 (July 2004) – English • Basis: Version PVSS II 3.0.1 under Microsoft Windows

This document is aimed at automation engineers, project engineers and developers seeking solutions in the fields of process visualization, instrumentation and control and Collaborative Manufacturing Management (CMM).

With its scalable, modular system design, PVSS II provides a powerful and future-proof base for every form of control-centre application. A client-server architecture is employed throughout the system, allowing multi-user access with simultaneous multilingual capability. The spectrum of possible configurations ranges from an autonomous single-PC system with minimum configuration to multi-computer applications for load distribution and whole multi-server clusters (distributed systems). Hot-standby server redundancy and full network redundancy satisfy even high availability demands.

The contents of this document were up to date at the time of writing and have been checked carefully. Ongoing development to improve the product further may mean that certain information is no longer valid and needs to be replaced by new data - please refer to the relevant customer information media and contact ETM support if in doubt. The document may contain errors both in content and form, and these cannot form the basis of any warranty claims. Many screenshots are based on one of several possible display modes of the Windows XP operating system (style: silver) or Windows 2000 - slight differences from other display modes are possible. To help the learning process, some modified screenshots (collages etc.) are used that cannot be seen directly in the product in the form shown.

If you find any errors in this or other documents of the PVSS II Online Help, please notify us in writing by e-mail to pvssdoku@etm.at or by a short FAX with the subject "DOKU" to +43 (0) 2682 – 741 – 107. Please direct any complaints, ideas and requests to the relevant contact department listed at the end of this document.

Trade marks, logos and product names of third parties, in particular the Microsoft Corporation, are subject to relevant separate copyright and trademark laws.

Authors: Beate Briss, Matthias Schagginger, Leo Knipp (all from ETM)
Document edition: Version 2.0 – English – Microsoft Windows platforms, July 2004

Contents

1 Introduction 5
1.1 What is PVSS II? 5
1.2 Terminology 6
1.3 Typography 7
2 Basic concepts 8
2.1 Architecture 8
2.2 Client-Server / Provider-Consumer 9
2.3 Communication, Event orientation 9
2.4 System, distribution, configurations 10
2.5 Datapoint model, process image 12
3 Installation 15
3.1 System requirements 15
3.2 Installation process 15
3.3 Opening the administration interface 17
3.4 Demonstration project 18
3.5 Licensing 19
3.6 Online Help 21
4 Project administration 22
4.1 Creating a project 22
4.2 Starting, stopping 23
4.3 Configuring a project 25
4.3.1 Console 25
4.3.2 System Management 26
4.3.3 Configuration files 26
4.4 Function buttons in Project Administration 27
4.5 After startup - First steps 27
5 Data model 29
5.1 Datapoints as information carriers 29
5.1.1 Datapoint types 29
5.1.2 Datapoints 30
5.1.3 Information at the datapoint element 31
5.2 Modelling datapoint types 32
5.3 Instancing 33
5.4 Settings for the datapoint element 34
5.5 Addressing 35
5.6 Functions at the device-oriented data object - configs 36
6 Creating datapoints 39
6.1 Creating custom datapoints in the PARA database editor 39
7 Graphical user interfaces - "Panels" 41

7.1 Creating process displays - the graphical editor 41
7.2 Simple drawing operations 42
7.3 The Property sheet 44
7.4 Making graphics properties dynamic (Simple Parameterization) 45
7.5 Scripting in the graphic 47
7.6 Using ready-made symbols for displaying data 48
7.7 The Preview in the graphical editor 50
8 Contact 51
8.1 Head office 51
8.2 Sales 51
8.3 Licensing 51
8.4 Training 51
8.5 Support/Engineering 51

1 Introduction

1.1 What is PVSS II?

"PVSS II" is the German abbreviation for "**Process visualization and control system II**", a software package designed for the field of automation engineering. Its main application is in the **operation and supervision of technical installations** using VDU workstations with full-graphics capability.

In addition to the **visualization of the current process states**, this application needs to be able to transfer **input values and commands** to the process and its control devices. The operator does this **interactively** using **mouse, keyboard** and other standard computer input devices, with the immediate response displayed on the screen. Other core functions of the software include **alerting** the operator when critical states occur or limits are exceeded, plus **archiving** of data for later display and analysis.

Such systems are usually called **control systems** or **visualization systems** or referred to by the acronyms SCADA or HMI. **SCADA** stands for *Supervisory Control And Data Acquisition* and sums up the essence of this software package particularly well. **HMI** stands for *Human Machine Interface*.

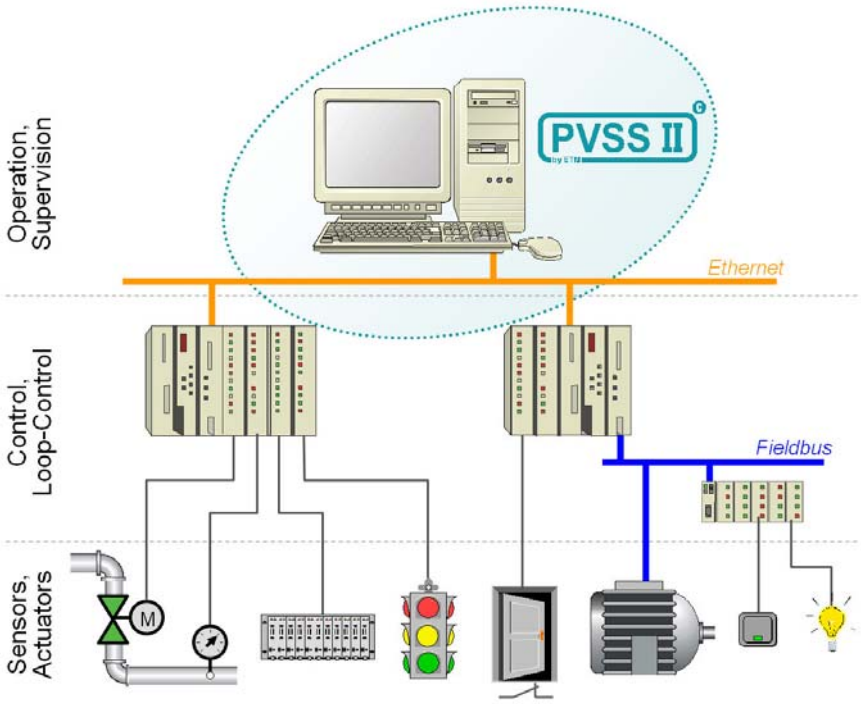


Fig. 1.1
Role of PVSS II within an automation system

So PVSS II is the **supervisory software for the control centre** or the **machine operating software**. **PC-based** servers and workstations are used as the hardware platform. Together with the control and regulation devices of the **automation platform** (PLC[1], DDC[2], RTU[3], ...) plus their sensors and I/O modules, these create a complete automation system.

[1] PLC…Programmable Logic Controller (industrial computer system for digital control and operation)

[2] DDC…*Direct Digital Control* (family of control devices in building services management)

[3] RTU…Remote Terminal Unit, Remote Telemetry Unit

1.2 Terminology

All technical and product-specific terms are explained when they are first used. A few particularly important abbreviations are also listed here:

Tab. 1.1
Main abbreviations and their meanings

Acronym	Full term	Meaning
PVSS II	Process visualization and control system (2nd generation)	Product name of a software package (control and visualization system) from ETM AG.
SCADA	Supervisory Control And Data Acquisition	A master system for gathering data on a process and for controlling/operating the process.
HMI	Human Machine Interface	The hardware and software for computer-based operation and monitoring of processes.
DPT	Datapoint type	Object definition (class) of a structured datapoint as a computerized representation of a real device. The individual datapoints (instances) are derived from the DPT. Thus the datapoint type is a sort of template.
DP	Datapoint	Structured device-oriented data object representing a real device within the control system. A datapoint contains one or more datapoint elements (process variables).
DPE	Datapoint element	An item of process information within a device-oriented datapoint. Every DPE corresponds to a value/state. In addition to holding the value, the datapoint element includes attributes giving the time stamp, quality information and the originator.
DPA	Datapoint attribute	In addition to the value actually represented (= process variable), each datapoint element contains a number of extra attributes - quality information ("status"), time stamp and originator as a minimum.
Config	Configuration	Control-system functions - "configs" - can be configured at each datapoint element, for example a range check, alert handling or archiving instruction.
UI	User Interface	The graphical user interfaces in PVSS II are also called UIs. "Native Vision", the runtime interface, is specifically often called the "UI".
GUI	Graphical User Interface	Generalized term for UI; see UI
API	Application Programming Interface	The API in PVSS II lets the user integrate his/her own functions and algorithms into PVSS II in the form of a new manager (see section 2).
GEDI	Graphical editor	Used in PVSS II for creating the process displays, symbols and dialog boxes.
PARA	Parameterization tool	Database editor for creating datapoint types and datapoints, and for configuring them.
VISION	Visualization module	Abbreviation for the "Native Vision" runtime user interface in PVSS II
PLC	Programmable Logic Controller	Industrial computer system for controlling and operating processes.

Acronym	Full term	Meaning
DDC	Direct Digital Control	Special class of control and operation systems for building automation.
RTU	Remote Terminal Unit, Remote Telemetry Unit	Telecontrol system
I/O	Input/Output	From the SCADA system viewpoint: data exchanged with external devices (e.g. PLC)
ASCII	American Standard Code for Information Interchange	Standardized protocol for storage and transmission of characters/text. In PVSS II, the acronym is also used to refer to the database import/export manager.

Tab. 1.1
Main abbreviations and their meanings

1.3 Typography

Explanation — Explanatory text with **bold formatting** to emphasize certain passages

Anglicisms — (*Programmable Logical Controller*)

Step-by-step — 

Hotkey — `[Ctrl] + [C]`

Buttons — OK, Next,

Menu, tab — Select the menu **File ⇨ Settings ⇨ Save tab**

Source code —
```
main(){ dpSet(….)... // comment
```

Paths, file names — *C:\pvss\GettingStarted_3.0*

Datapoint name — **`V2.state.open`**

Online Help reference —  **{Control ⇨ Control Graphic ⇨ Checkbox functions}**

Module name — GEDI, PARA, VISION, NG, NV, EV, Event Manager, Data, ...

Tip box —  Tip

Caution box —  Caution

In-depth info —  More detailed information at the end of that section

2 Basic concepts

2.1 Architecture

PVSS II has a **highly modular design**. The functions required are handled by **functional modules specifically** created for different tasks. These modules are called "**managers**" in PVSS II; they are also separate processes in software.

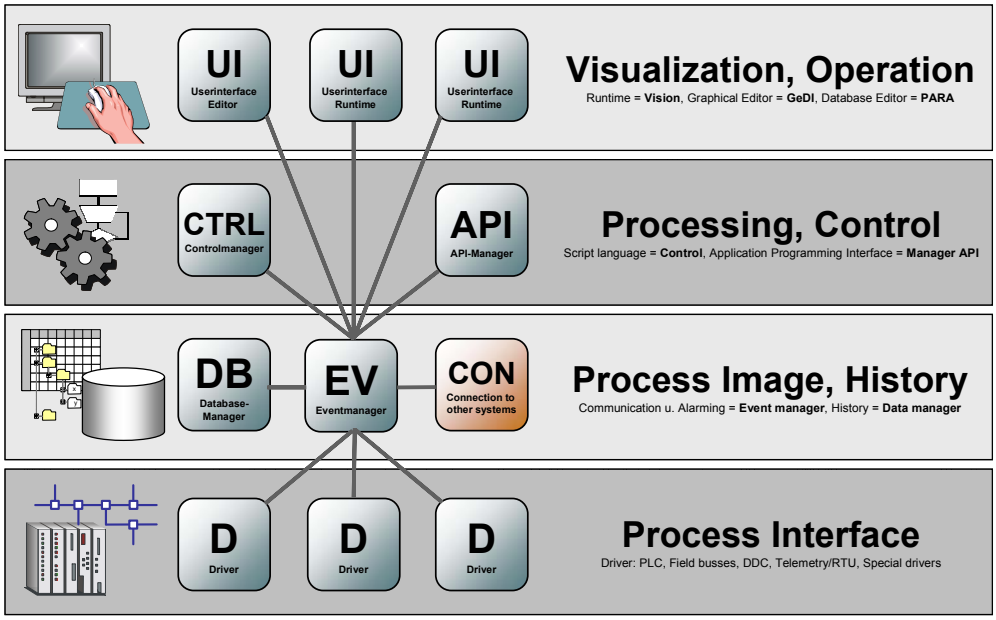


Fig. 2.1
A PVSS II system consists of function-specific managers known as managers

The roles of the most important managers, shown in figure Fig. 2.1, are explained briefly below. This diagram shows just **a simple configuration**, which may be far larger in practice. In fact there are also a number of other managers in addition to those mentioned above, which are not dealt with further here.

Process interface

The process interface modules, referred to as **drivers** (**D**) in PVSS II, form the lowest level of a PVSS II system. These are special programs that handle **communication with the process control and field level**. As the possible forms of communication with PLCs or telecontrol nodes vary widely, there are **a number of different drivers** to select from. The **PLC** employed and the associated communications bus therefore decides which PVSS II driver shall be used. Put extremely simply, the driver is a **module that converts a specific protocol** into the form of communications used internally by PVSS II. The driver reads online states, measured values or counter readings from the field, and passes commands and setpoint values back down to the controllers.

Process image, history

The **central processing unit** in PVSS II is called Event Manager (EV). This unit holds the current **image of all process variables** in the memory. Every other function unit (*Manager*), which want to access the data, receives these data from the process image of the Event Manager and do not have to communicate directly with a controller. Vice versa a command from a control station is set as a value change in the process image of the Event Manager in the first instance. Afterwards the responsible driver forwards the value to the specific target device (e.g. PLC) automatically.

The Event Manager is a central data distributor, the communication center for PVSS II. Additionally this manager executes also the **alert handling** and is in a position to make different **calculation functions** autonomically.

The Data Manager (DB) sits to the side of the Event Manager. It constitutes the link to the database. It not only handles the **parameterization data of an application** to be saved in such a database, but also the **archiving** of **value changes** or **alerts**. When a user wants to retrieve historical (archived) data at a later date, then the Data Manager also deals with this request and not the database itself.

Processing, Control

PVSS II includes numerous options for implementing **one's own algorithms** and **processing routines**. The two main methods are the internal language Control (CTRL) and the API (*Application Programming Interface*) general programming interface.

Control is an extremely powerful scripting language. The code is processed interpretively so does not need compiling[1]. It has largely the same syntax as ANSI-C[2], with some simplifying modifications. It is an advanced **procedural higher-level language** that uses multithreading[3]. The language provides a comprehensive function library for tasks in process control and visualization. Control can be used as a **self-contained process** (CONTROL Manager), for animation and **user-interface configuration** (UI Manager) or for standardized **data-object based processing** (Event Manager).

The **API** (PVSS API) is the most powerful form in which to add extra functions. It is implemented as a **C++ class library** and lets the software developer implement custom functions as **additional self-contained managers** (forecasting system, simulation, design tools, proprietary databases, etc.).

Visualization, operation

The User Interface (UI) Managers form the interface with the user. These include a **graphical editor** (GEDI), a **database editor** (PARA) and the general **user interface** of the application (Native Vision, UI). In the User Interface, values are displayed, commands issues or alerts tracked in the List of alerts. **Trends** and **Reports** are also normally included in the UI. In PVSS, the user interaction software runs completely separately from the processing executing in the background - it merely provides a window on the live data from the process image or the archived data in the history.

2.2 Client-Server / Provider-Consumer

The **individual managers interact** as in a **true client-server architecture**. This implies that the servers execute their processing tasks and provide data **independently** of the client. In this model, servers are the **information providers**.

A client, put rather simply, is the **recipient or consumer** of information, which it receives from the server. This is often described as a **Provider-Consumer relationship**.

This role demarcation is not confined just to runtime visualization (NV, UI) and the Event Manager in PVSS II; in fact all communications relationships between managers follow this principle.

2.3 Communication, Event orientation

Data **processing** and **communication between the individual processes** (managers) is normally performed purely on an **event-oriented** basis in PVSS II. This means that spontaneous (immediate) processing or

1 Conversion process in which a program command is converted from source code into the machine code executable by the processor.

2 "C"... internationally standardized and extremely widely established higher-level programming language

3 Quasi-parallel processing of individual programs; the system itself performs processing monitoring.

transfer of a value occurs **if and only if it changes.** Conversely, in steady-state operation with no changes in values, there is neither communications nor processing load.

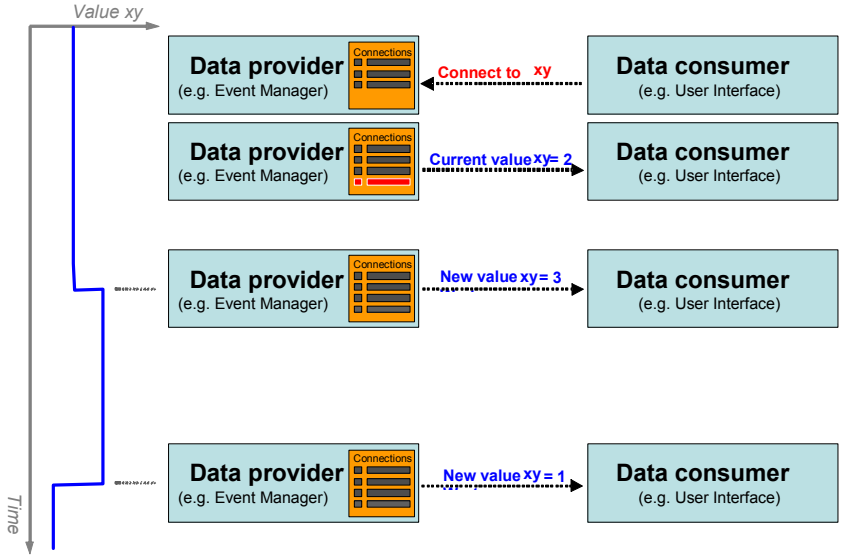


Fig. 2.2
Time sequence of event-oriented communication (Connect-Callback)

The system works **very efficiently** and is active "**only on demand**". An application programmer is provided with all the necessary structures to do this: in accordance with the "Provider-Consumer" communication role described in section 2.2, functions are provided which a user software module or interface (*consumer*) can use to **register** with (*connect* to) changes in value from a data source (*provider*). Once registered (*connected*), every new value is automatically transferred from the data provider to the consumer and input to the specified processing.

The individual managers communicate via a **TCP/IP message interface**. This **reliable and established** form of communication enables data transfer even between different computers and operating systems. The **global TCP/IP standard** guarantees maximum **reliability**, **compatibility** and **performance**.

2.4 System, distribution, configurations

A structure made up of **one** Event Manager, **one** Data Manager and various other managers is called a **system** in PVSS II. An Event and Data Manager alone already form an operational system, usually with at least one driver (D).

All other managers e.g. a user interface (UI) or a Control Manager (CTRL), are **only started when they are needed**. This enables **scaling** of the system according to need. Managers can be **started** and **stopped** entirely during online operation **without restarting the whole package**.

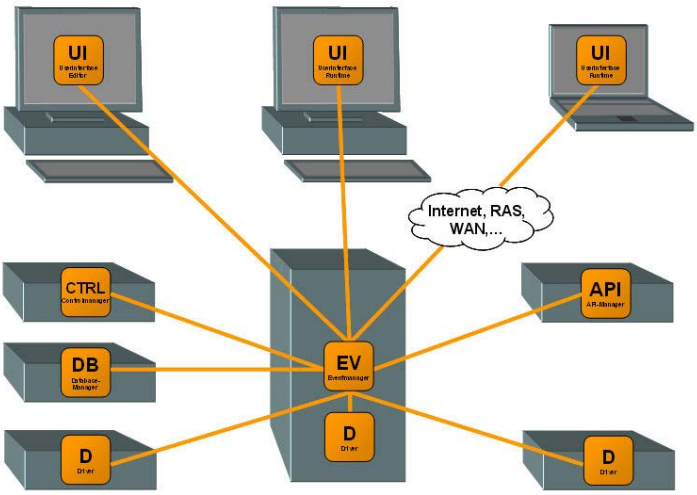


Fig. 2.3
Distribution of a PVSS II system over several computers; communication: TCP/IP

Furthermore, not just one but **several instances** of a manager can also be started if required for all manager types (UI, CTRL, D, API, ...). Thus a number of user interfaces or drivers can be run from one Event Manager for example. There is **just one** Event Manager and **one** Data Manager **per system**.

The modular design and neutral, TCP/IP-based communication mean that a PVSS system can be **distributed across a number of computers**. This allows

- **demorcation** of functions
- **load distribution**
- operation across **platform boundaries**

This means that the customary division between workstation (UI Client) and server (EV, DM, ...) is taken a step further: Drivers, Control Managers or API´s can also run on different computers.

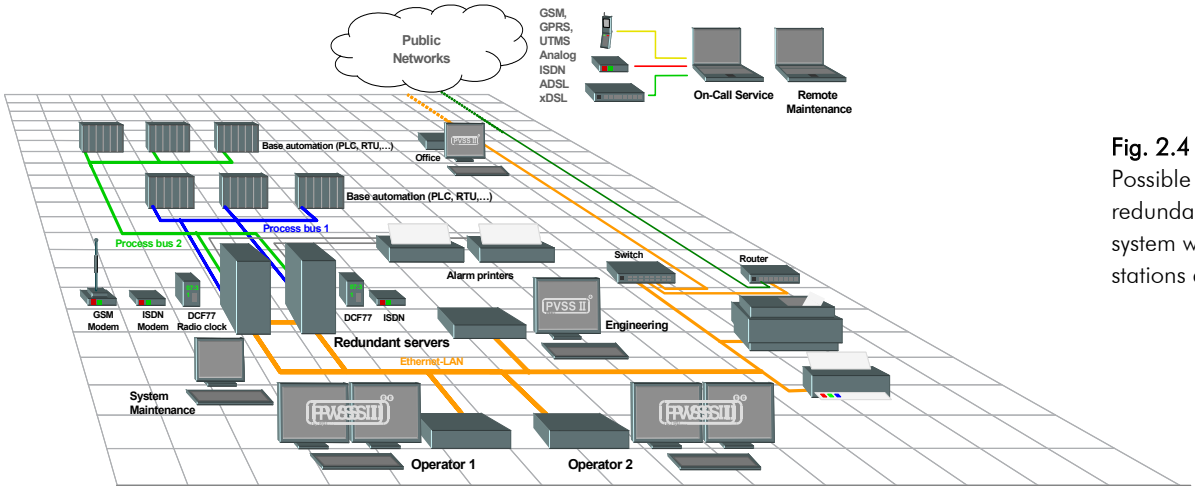


Fig. 2.4
Possible configuration: redundant client-server-system with 4 operating stations and home console

The distribution of the managers in a system to different computers is not confined to one operating system platform. Many users employ Windows (2000, XP) for the user interfaces for example, while the SCADA server runs under LINUX.

Redundancy, distributed systems

PVSS II offers the option of running all **server processes** on two computers in the form of **hot-standby redundancy** (see Online Help **{Special functions ⇨ Redundancy}**). With PVSS II it is also possible to interconnect a number of autonomous systems into an overall system (**multiserver architecture** / **distributed system**) simply and efficiently (see Online Help **{Special functions ⇨ Distributed systems}**). These "higher value" configurations are not included in this introduction to PVSS II however.

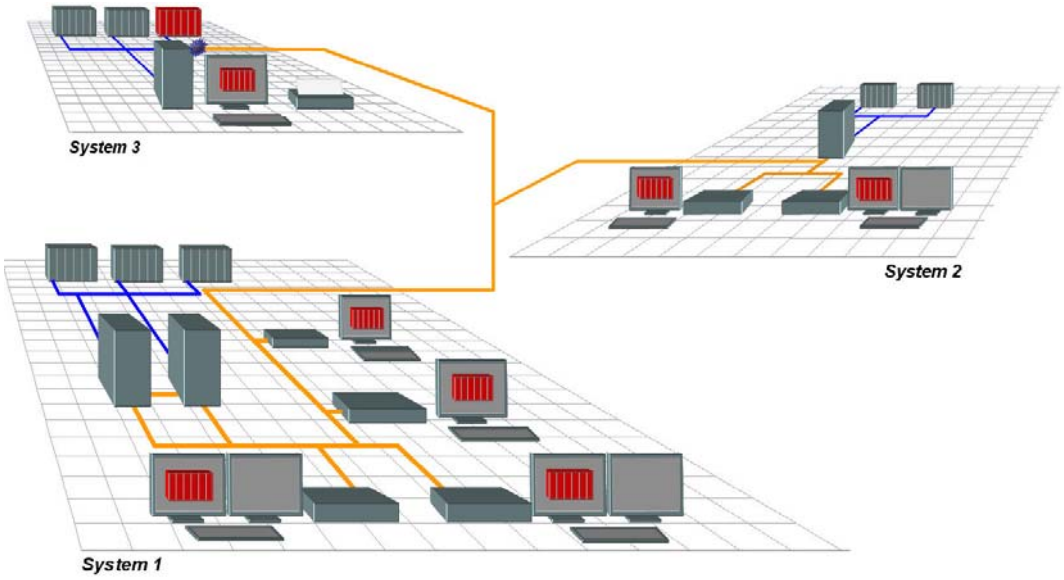
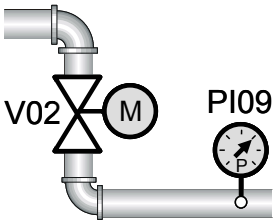


Fig. 2.5
Hot-standby redundancy and distributed systems (multi-server architecture)

2.5 Datapoint model, process image

The **variables** of the **process** to be controlled and monitored must also find their way into the software at the control desk. Every **logic state**, every **measured value** or **setpoint value** must **correspond** to a sort of **variable** that **represents** this value within the system.



Datapoint	Description	Units	Value
`V02.state.closed`	Valve V02 response, Final position closed	-	TRUE
`V02.cmd.open`	Valve V02 command Open	-	FALSE
…			
`PI09.value`	Pressure reading P109 actual value	bar	2.74
…			

Fig. 2.6
Mapping of states, input values and measured values onto datapoints (process variables)

These **variables** of the **process image** are called "**datapoints**" in PVSS II. Many different names are used for these information carriers depending on the product or region (tag, process variable, PV, item, point, I/O point, etc.).

While standard SCADA systems assign a separate datapoint to every individual process variable, PVSS II takes a more modern approach: nearly all information in the process belongs logically to an entity of varying complexity: **a device**.

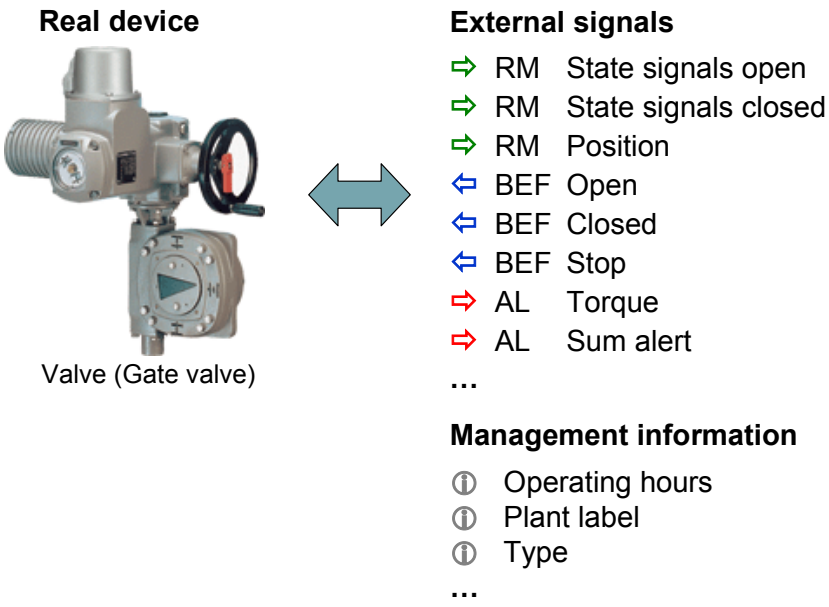


Fig. 2.7
Signals from a real device

Experience has shown that the number of **related items of information of such a device** typically lies between 4 and 30. Intelligent modules such as digital controllers, function modules, robots etc. can well exceed this number.

Instead of transferring these otherwise logically, closely related items of information to independent variables, which would then make them completely separate from each other, PVSS II defines **structured, device-oriented datapoints** instead. The datapoints are defined in a sort of **tree structure** which can have as many branch levels as required.

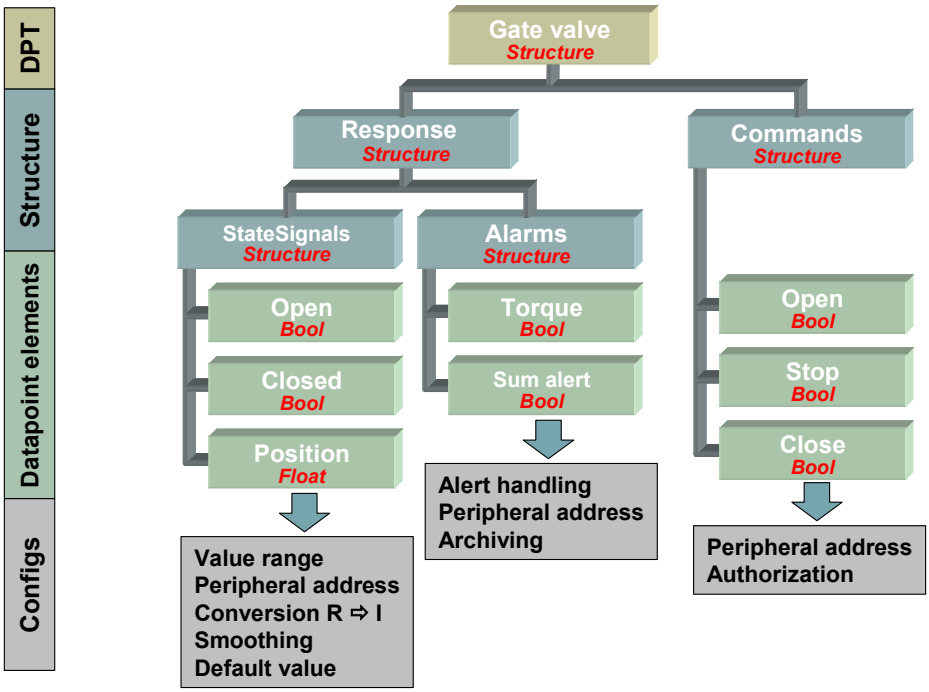


Fig. 2.8
Structured datapoint as a representation of a real device (*device orientation*)

The **values of the actual process variables** are saved on **datapoint elements**, the outer leaves of this tree structure. Each process variable therefore corresponds to one datapoint **element** within a datapoint. In addition, the tree structure can have as many nodes as necessary for clear organization of the data.

Each datapoint element is **addressed** individually via the **name chain within the structure**. The status message "**Open**" in the above example can be addressed in full for instance by:

```
gate_valve.response.SateSignals.Open
```

Of course in practice one would use shorter names to save writing; nevertheless PVSS II allows character strings of up to almost any length.

In addition to the **name convention** and a **storage location** for the actual value, certain **process-control functions** can be defined at the datapoint, for example a range check, an alert handling procedure or a statistical computation rule. Such **functions defined at the datapoint element** are called "**configs**" in PVSS II. Only those configs that are actually needed at the datapoint element concerned are defined.

Datapoint type and datapoint

Thus the user can create a **suitable datapoint type for each real device type** (actuator, valve, stirrer, regulator, intruder sensor, etc.). **A datapoint** is then derived from this datapoint type (a kind of **template**) **for each real device**. In object-oriented software engineering, the datapoint type would be called a "class" and the representation of an individual device (i.e. the datapoint) an "instance".

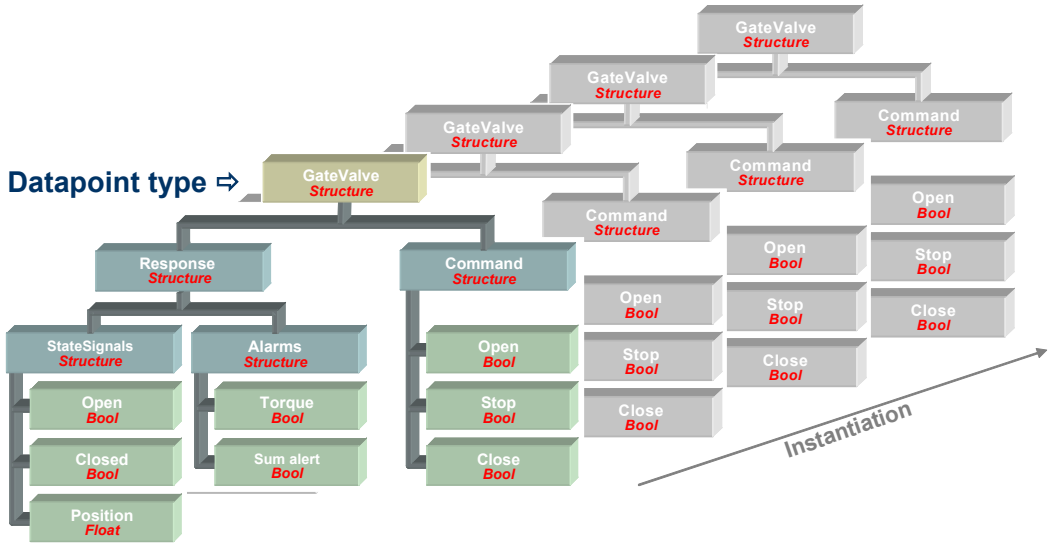


Fig. 2.9
Datapoints slide valve 1 to 3 as instances of the datapoint type slide valve

Thus **creating** and sometimes also **configuring** a **large number of process variables** representing a device involves just a **single operation**. Pre-defined datapoint types representing a module (e.g. an operating-time counter) can then be taken **as a whole** and **used in a new datapoint type.** New ultra-efficient engineering opportunities present themselves using these **hierarchically structured datapoints** ("type-in-type").

Changes to the datapoint type are also applied automatically to the datapoints (instances).

3 Installation

The system requirements and installation procedure described below only apply to **version 3.0.1** of **PVSS II** running under **Windows 2000** or **XP**. For instructions for installing under Linux, please consult the Online Help **{Getting Started ⇨ Installation Linux}**. The target computer specification with regard to speed, RAM size and hard-disk capacity only applies to the applications and projects described in this document.

3.1 System requirements

Check that your computer meets the **system requirements**[1] listed here **before installing** the software on it: IBM compatible PC (Intel x86 architecture):

	Minimum	Recommended
Processor: Intel Pentium or equivalent	P II 350 MHz	P IV 2.4 GHz
RAM	256 MB	512 MB
Virtual RAM[2] (in addition to RAM on HDD)	384 MB	512 MB
Hard disk (spare capacity)[3]	500 MB	1 GB
Monitor and graphics card (TrueColor)	1024 x 768	1280 x 1024
2-button mouse, US English keyboard	✓	✓
CD-ROM drive for installation	✓	✓
Standard network card (10/100)		✓
Microsoft Windows 2000 Professional	SP2	SP3
Microsoft Windows XP Professional/Home	SP1	SP1
Microsoft Internet-Explorer (V 5.5 or above)	✓	✓
Local administrator rights (for installation)	✓	✓
Local main user rights (for operation)	✓	✓

Tab. 3.1
System requirements for applications in this introductory guide

Internet e-mail access is recommended as the simplest way of handling the licensing process. If this is not available then phone or fax can be used.

3.2 Installation process

The installation described here includes all components required to run the examples in this manual on **a single computer,** and for taking one's own first project engineering steps. The basic components for

1 The hardware requirements used in real process applications depend heavily on the project size and the rate of change of the process variables. Use high-quality robust hardware with suitable specifications such as redundant PSUs or RAID hard disks. PVSS II supports and benefits significantly from dual- and multiprocessor operation. As always for RAM, CPU and HDD, the more the merrier.

2 It is recommended to set the size of the virtual RAM to a value between one and two times the physical RAM. Select a value that is an integer multiple of 32 MB and specify the same value for the initial size and the maximum size (no dynamic variation in virtual memory)

3 in addition to the virtual RAM requirements

operation and parameterization and three example projects[1] are installed locally (**single-user system**). Please consult the Online Help **{Installation}** for all other installation options.

The ***Readme.txt*** file in the root directory of the installation source (CD-ROM, network) contains both an installation guide and a list of the latest changes and additions to the Online Help.

Before starting the installation, make sure that your system meets all the **system requirements**. In particular, check that **local administrator rights** are granted for the operating system under the current login.

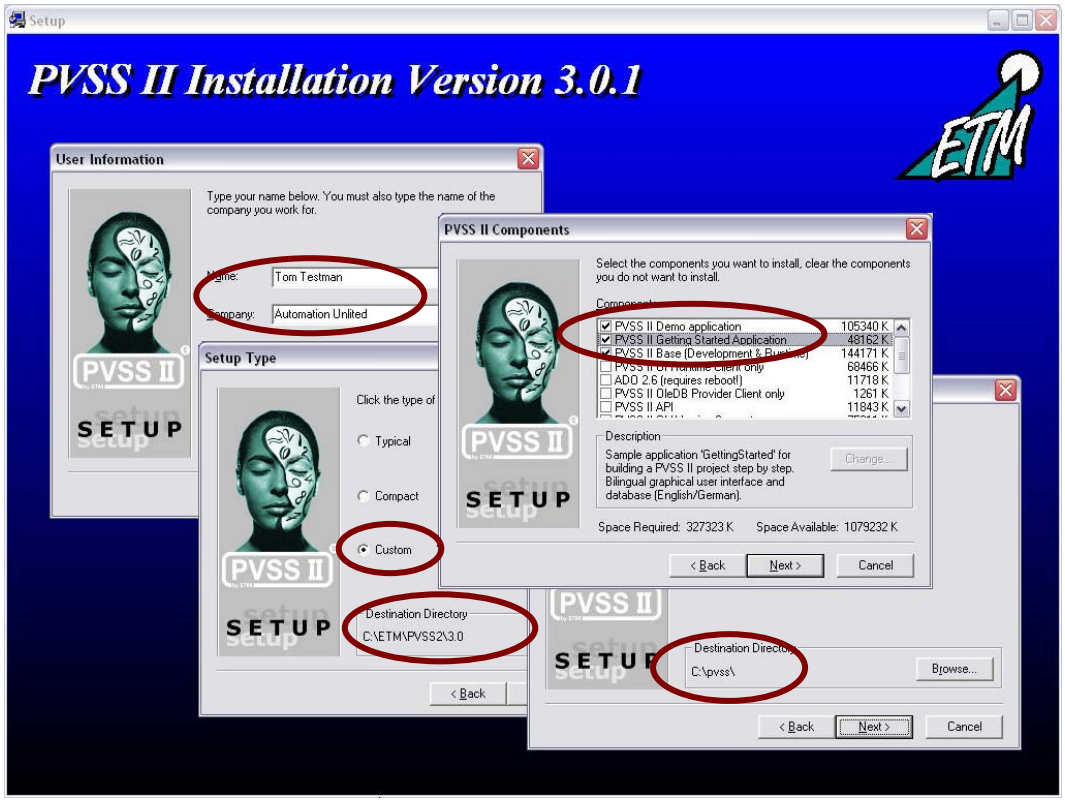


Fig. 3.1
Installation options to be selected for this guide

Close any programs that may be running and insert the CD-ROM "PVSS II Version 3.0.1 / Windows" (or above) in your computer drive. Wait until the **Setup program** autostart option opens. Alternatively you can start the program manually by **running the file *setup.exe*** in the root directory of the installation source. The **Installation dialog box** guides you through the installation process.

Running other programs during the PVSS II installation phase can result in an incomplete or incorrect installation. Also check that there are no programs running in the background. Furthermore, some programs such as MS Outlook slow down the installation software dramatically.

1 two example projects (DemoApplication_3.0 und GettingStarted_3.0) and an empty project (Demo_3.0)

Please perform each step of the **Installation wizard** as described below. Always use the default paths suggested wherever possible, because all subsequent advice and instructions are based on these. Use the default settings except for those options mentioned explicitly here, and click on the Next button

1. Enter your full **name** and your **company name**
2. Select the "**user" installation option**
3. Select the 3 top options from the component selection (**PVSS II DemoApplication, PVSS II Getting Started Application** and **PVSS II Basis**); do not change the other options.
4. Confirm the **directory where you want the files installed** (*C:\Program files\ETM\PVSS2\3.0*)
5. Confirm the **installation directories for the applications** (*C:\pvss*)
6. Confirm all subsequent prompts with Next.

PVSS II tells you that the installation process has finished once correct installation is complete. Depending on which system components were already installed on the installation computer, **it may be necessary to restart the computer**; the Setup program tells you if this is the case.

Otherwise you can just close this last dialog box and open the **PVSS II administration programs** via the standard startup environment of your computer system. You can find further details in the Installation section of the Online Help **{Getting Started ⇨ Windows installation, Installation}**.

Should you need to remove PVSS II from your system, use the uninstall program supplied with the software, which you can run via **Start ⇨ Programs ⇨ PVSS II 3.0 ⇨ "Uninstall PVSS II 3.0".** Alternatively you can use this option under **Control Panel ⇨ Add or Remove Programs**.

3.3 Opening the administration interface

PVSS II can now be started using the operating system **Startup cascade** via the path **Start ⇨ Programs ⇨ PVSS II 3.0 ⇨ PVSS Project Administration**.

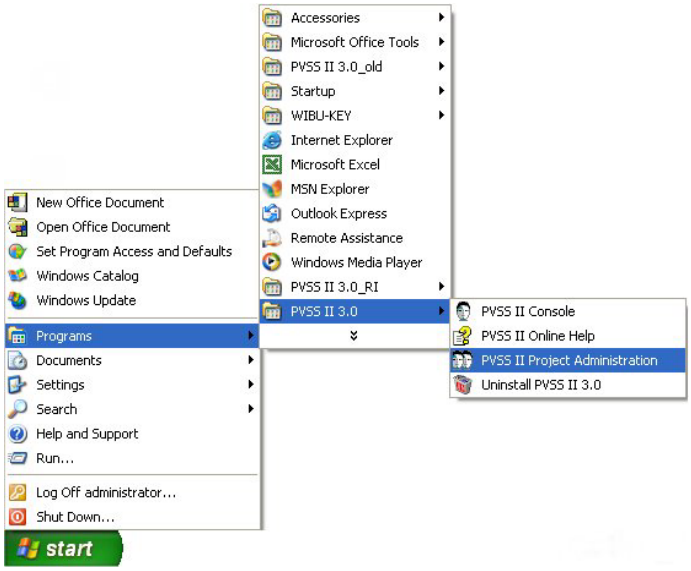


Fig. 3.2
Starting the "Project Administration" user interface of PVSS II

After startup, the **list** of all **projects installed** on the current computer is immediately displayed. In PVSS II a "project" means an executable module (user interfaces, scripts, datapoints, etc.) created for an explicit application. **Several of such projects can be managed on one computer** using a PVSS II parameterization workstation, but normally only **one individual project is ever open** at any one time. Such projects are sometimes also called "applications".

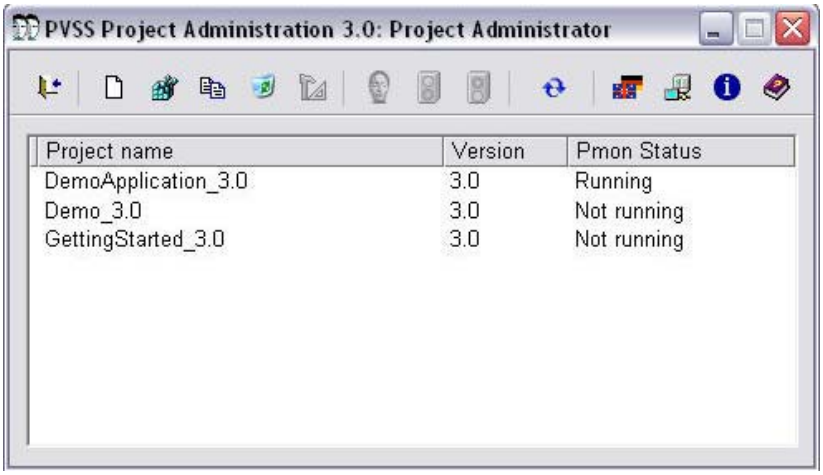


Fig. 3.3
Project administration
Administration user-interface for creating new projects

Selecting a project from the list and clicking on the icon displays the selected project in the CONSOLE **startup environment**. It is then ready to be **started**.

3.4 Demonstration project

A project called **DemoApplication_3.0** is also installed on the computer at the same time as the installation. This is a **standard demonstration project** as also used by sales staff and trainers. The aim of it is to demonstrate to the user a range of **possible user-interface layouts** and **navigation options**. The Online Help dedicates a chapter to this application **{PVSS DemoApplication}**.

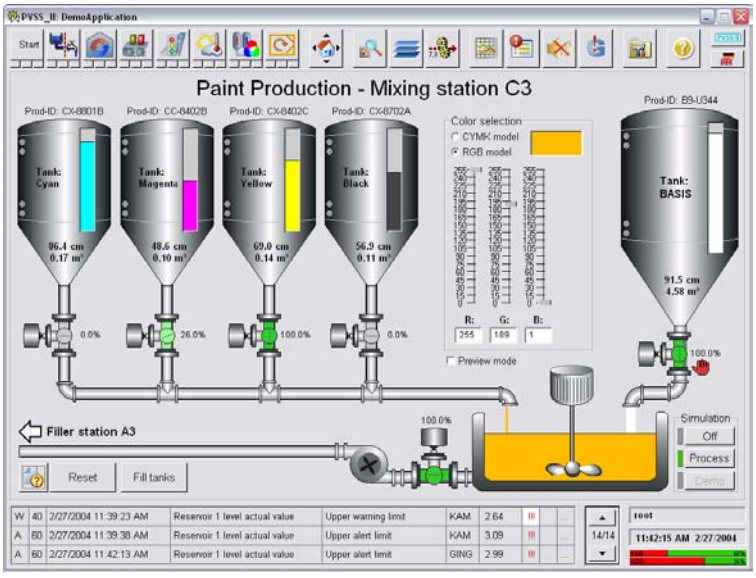


Fig. 3.4
DemoApplication_3.0 - a demonstration project for Sales and Training

It is highly recommended to take a closer look at this DemoApplication_3.0 from the viewpoint of the end user, in addition to **your own first project-engineering steps** based on this document. Advanced users can look at the DemoApplication_3.0 to find out about possible implementation steps that can be adopted in their own applications. The DemoApplication_3.0 project is not dealt with further in this document.

3.5 Licensing

The PVSS II software package requires a **manufacturer's licence**. To protect the product from unauthorized duplication, **special activation** is required for it to run. This is performed using either a **hardware-dependent enable code** (file) or **hardware protection** (USB/parallel port dongle). This authorization defines both the explicit computer, on which the software package is authorized to run, and the system configuration and size approved for this.

To activate the product for use, proceed as follows: in Project administration click on the  icon in the toolbar.

Automatic licensing by e-mail

If you have an e-mail compatible connection on the computer to be used for the PVSS II installation, then complete the dialog box as shown in Fig. 3.5 . Enter the name of your mail-out server (SMTP server) and add **additional contact details and the functions you require** in the text part of the e-mail. Also enter the address to which you want the granted licence to be sent (reply address). Then press the button containing the green checkmark (tick) to send off the licence request. This direct option is **only** practicable if your company is **already a partner of ETM**.

Manual licence request

By opening the dialog box mentioned above, the system determines the **hardware-specific code of your computer**. It is displayed in the top right field 'Hardware code:' and can be saved as a text file using the  button. If you confirmed the default directory at installation, then this file can be found in the root directory of the version directory.

C:\Program files\ETM\PVSS2\3.0\hw.txt or *<PVSS_PATH>\hw.txt*

If you are making a manual activation request for the computer concerned, then close this dialog box again. You can send the aforementioned file directly to ETM, or another authorized licensing centre, by e-mail, fax or letter, or read out its contents over the phone.

Details of a **commercial product order are** also required in addition to the hardware code. This order specifies the **required software configuration** (number of datapoints, workstations, features, add-ons, etc.), based on which an **enable code suitable for** the computer specified by you is generated. Please contact your supplier or the ETM Sales department if you have any queries relating to the commercial product order.

This enable code is passed to you in the form of a **specially formatted text** file. Please also save this file in the version directory (as above).

C:\Program files\ETM\PVSS2\3.0\shield or *<PVSS_PATH>\shield*

Make sure that this is the only file in this directory with the name "**shield**". Whether the file does or does not (old version) have a *.txt extension is irrelevant. Just make sure that there is only one file with the filename ***shield***. You can do this either by renaming a supplied file ***shield.txt*** as ***shield*** (without the .txt extension) or retaining the filename ***shield.txt*** and removing any other similarly spelt files ***shield*.***.

If any changes are made to the hardware configuration of the computer, the ***shield*** product activation file may lose its validity. Please contact ETM in this case.

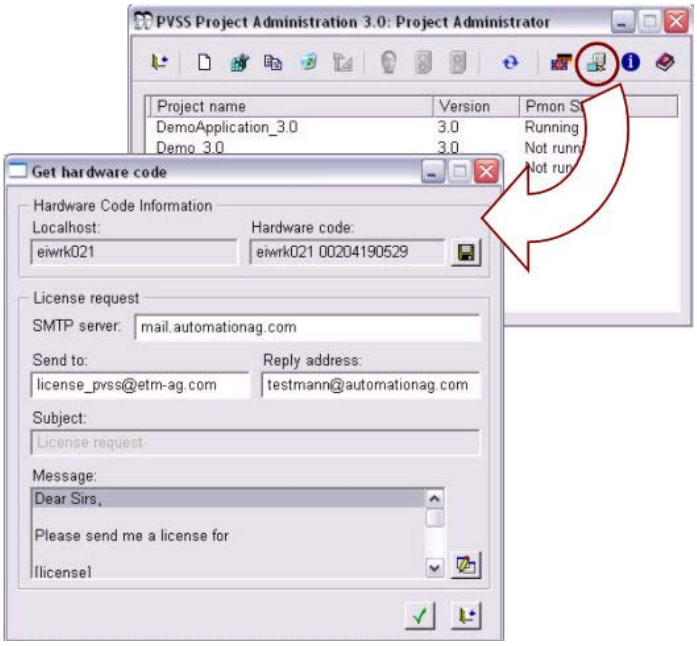


Fig. 3.5
Generating the hardware code for the computer, and automatic licence request by e-mail

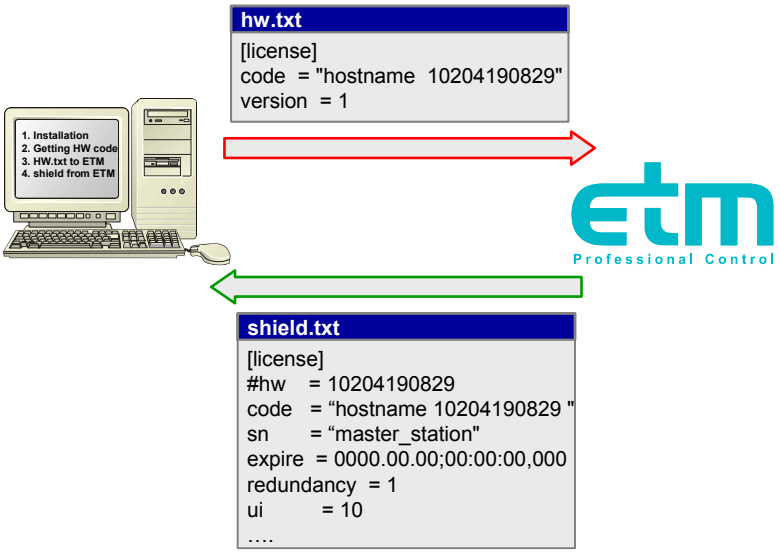


Fig. 3.6
Procedure for technical licensing of PVSS II - hardware-dependent enable code ("shield")

If you are not yet an ETM customer but would still like to learn more about PVSS II using the Demo CD, please contact the Sales department. If necessary they can grant you a limited-period evaluation licence.

If you try to run PVSS II without a valid licence, then the application warns you of this when it starts up. The system will automatically shutdown without warning after 30min maximum.

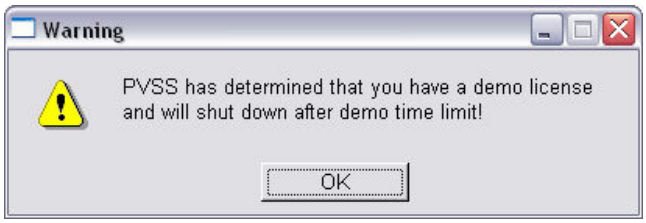


Fig. 3.7
Automatic warning if there is no licence

3.6 Online Help

PVSS II is supplied with a comprehensive **Online Help** facility. This was installed automatically with the installation described above. You can access the Help system via the Windows Startup cascade **{Start ⇨ Programs ⇨ PVSS II 3.0 ⇨ PVSS Online Help}.**

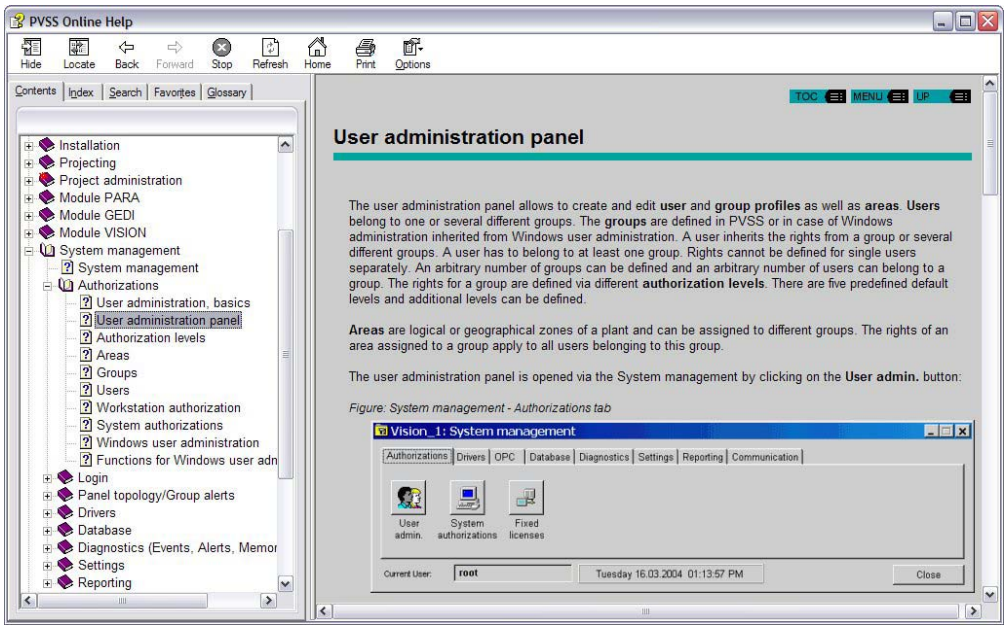


Fig. 3.8
Online Help system with topic tree, fulltext search and Index

The Help facility can also be opened from the various graphical user interfaces via the **"?"** menu option, the icon or buttons bearing the text Help. A specific topic is flashed up depending on the context.

The Online Help contains **several thousand pages** and handles the different operating system platforms in parallel. To make it easier for beginners, this manual highlights relevant **references to the corresponding Help pages** in the form **{Chapter ⇨ Section}.**

Also read the *Readme.txt* file in the installation source directory - in addition to details of the installation process, this contains any recent changes to the Online Help made after going to press.

4 Project administration

In PVSS II a "**project**" means an **executable module** for fulfilling a process control or visualization task. Normally **just one project** runs on **one computer**. Additional to the programs and basic components provided with the installation, **the project contains all specific parameterizations for a task**. These basically include:

- **Datapoint types** (definition of devices)
- **Datapoints** (variables of the process image / representation of devices)
- **Panels** (process displays, dialog boxes, symbols)
- **Facility-specific processing routines** (scripts, clock timers, recipes etc.)
- **Configurations** (archive settings, alerting, trends, etc.)

4.1 Creating a project

In order to be able to work with PVSS II, **at least one project** must exist on the computer. Three projects have already been created and installed ready for use during the above installation (see Fig. 4.1). We shall create **our own new project**, however, to be used for our own personal project-engineering steps, by following the instructions below:

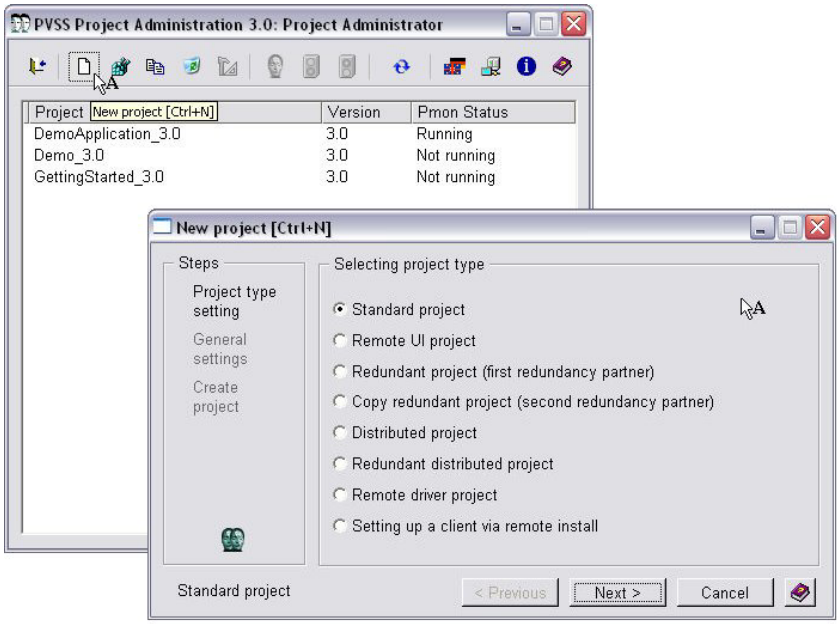


Fig. 4.1
Creating a new project using the Project Administration wizard

1. Open the Project Administration program
   (**Start ⇨ Programs ⇨ PVSS II 3.0 ⇨ PVSS II Project Administration**)
2. Click with the mouse on the icon ("**New**") or use the `[Ctrl]+[N]` hotkeys.
3. In the wizard, select the "**Standard project**" installation option and click on Next
4. In the "**Project name:**" field enter a project name e.g. the character string "**myGettingStarted**"
5. Use the mouse to select the "**English - UK**" option as the project language.

6. Select the installation directory for the project using the directory selector or enter the path by hand: **C:\pvss\;** then click on Next
7. Confirm the dialog box that then opens with OK **to create the project** – wait while the database is generated (this can take some time depending on the choice of language)

Every form of character string in PVSS II is **case sensitive**. Ensure that the name that you give always matches exactly the spelling and case given here (applies to project names, names of graphics objects and panels, variable names and much more).

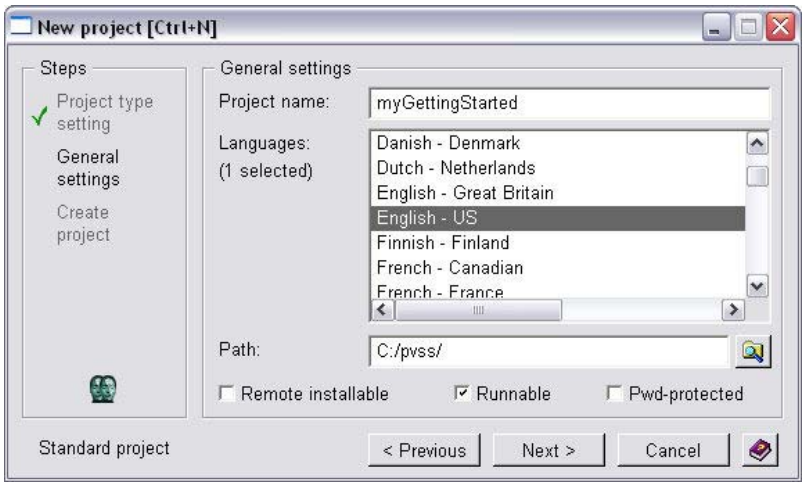


Fig. 4.2
Selecting the project language(s) and the installation directory

If you move the mouse over toolbars and buttons, then context-sensitive texts explaining the functions are displayed (tool tips).

Once the project has been created, the new project is displayed with its name in the Project Administration program. It is now ready to be **started** for the first time (see section 4.2)

Project name	Version	Pmon Status
DemoApplication_3.0	3.0	not running
Demo_3.0	3.0	not running
GettingStarted_3.0	3.0	not running
myGettingStarted	3.0	not running

Fig. 4.3
Project list in the Project Administration module after creating "*myGettingStarted*"

4.2 Starting, stopping

As soon as the required project has been created on the computer, the Project Administration program is no longer needed. So select the required project with the mouse and then switch to the Console program by clicking on the icon. Alternatively you can open the Console program via the context menu (`[right mouse-click]` on the project line) and start the project immediately.

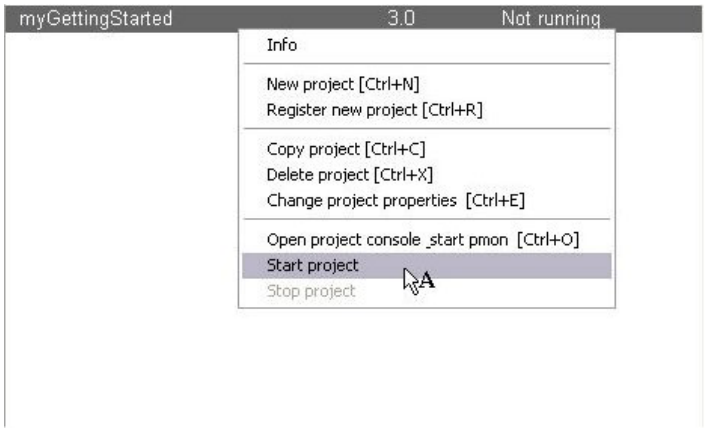


Fig. 4.4
Starting a project via the context menu in Project Administration

A project to be started up individually can also be selected in the Console without needing to use the Project administration program ("Project" combobox).

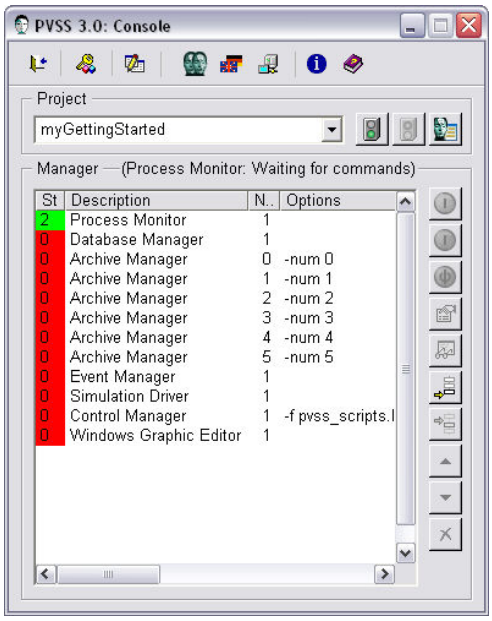


Fig. 4.5
Console as control centre for **one** project - Manager list with operating status

The Console **is the control centre of a project**. **Those managers** that need to run in order to operate the facility are specified here. Their **startup order** is also defined here, as are the necessary measures for **process monitoring**.

Clicking on the **green traffic-light icon**  **starts** the selected project. The startup procedure, which runs **sequentially**, can be viewed in the Console.

A project can also be **closed** from the Console by clicking on the **red traffic-light icon**. It is **shut down** in the **reverse order** from the startup procedure. The coloured background of the first column always shows the **current operating states** of the manager concerned.

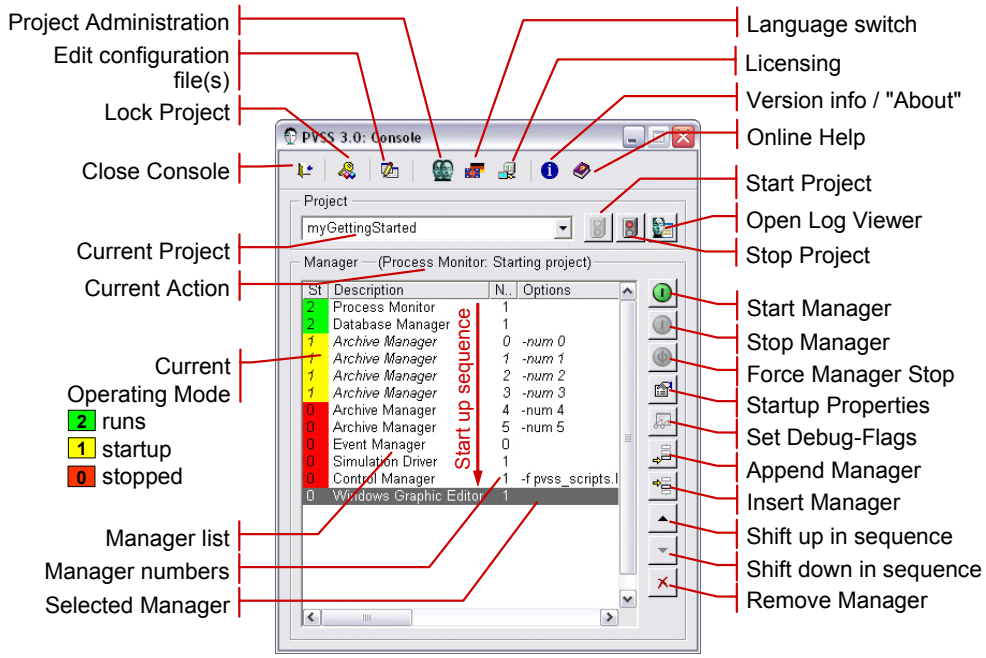


Fig. 4.6
Displays and function buttons in the Console

Rather than **starting** a project in full, **each of the listed managers** could also be **started or stopped individually manually**. This is done either via the context menu `[right mouse-click]` in the manager list or using the buttons arranged vertically on the right of the list. Note that some of the managers can only be started (manually) if the Data and Event Manager are already running.

The Console works **independently of the actual PVSS II system**. If just the Console is closed, then the managers started from it still continue to run. As soon as the Console is re-opened, it displays again the current operating states of the managers. Thus during normal operation of the facility, it can remain closed just like the Project Administration module.

4.3 Configuring a project

In addition to defining datapoints and drawing process displays, a **number of configurations** need to be made in a PVSS II project. This includes parameterization of the process interface modules (drivers), configuring distributed systems and setting the colour display for the alert panel. Such settings are performed **at three basic locations**: Console, System Management and Configuration files:

4.3.1 Console

The managers belonging to a project are chosen in the Console. This is done simply using the buttons for adding managers  . The dialog box that opens can be used to select a manager and define its startup properties.

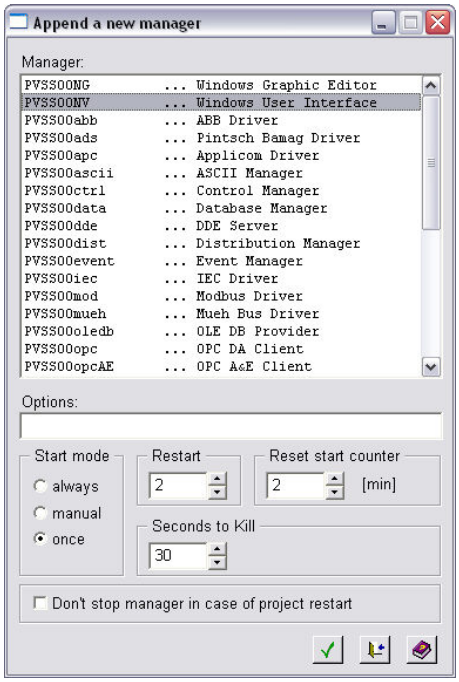


Fig. 4.7
Adding a new manager to the project and specifying the startup properties

Alternatively, the **startup properties of a manager** can also be defined directly in the **Manager list of the Console** by **double-clicking on the relevant line** or clicking on the icon. The possible options are shown in the Online Help **{Reference tables ⇨ Manager options}**.

4.3.2 System Management

The System Management module is essentially a **management centre** for managing a huge range of settings. It can be opened from the PARA database editor, the GEDI graphical editor or even from the NATIVE VISION runtime user interface via the **icon**.

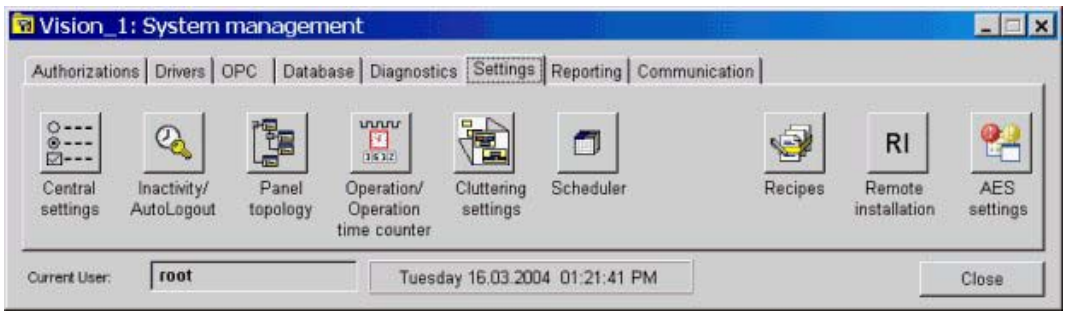


Fig. 4.8
System Management: Settings for the project and various tools

4.3.3 Configuration files

PVSS II uses a number of configuration files for making certain **project settings**. This configuration facility is used extensively, especially in conjunction with **process interface modules** (drivers). The main configuration file is located in the project directory in the path

C:\pvss\myGettingStarted\config

4.4 Function buttons in Project Administration

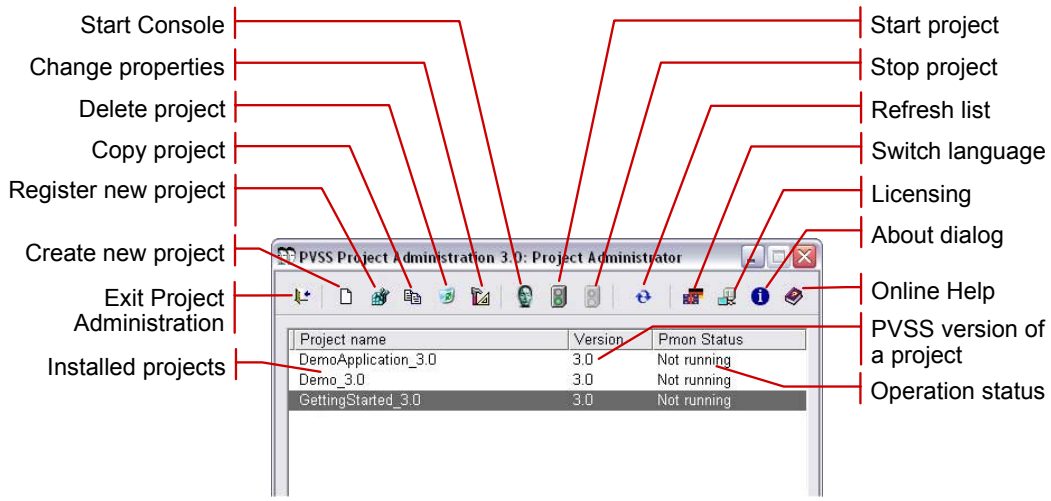


Fig. 4.9
Functions in Project Administration

4.5 After startup - First steps

With a new project, a **number of predefined manager entries** appear in the Console. We will be adding to these later, but to start with this environment provides everything we need. After **starting** the project by a `[left mouse-click]` on the green traffic-light icon at the top to the right of the project name (Console), and once the **startup procedure is complete**, your user interface should look something like this:

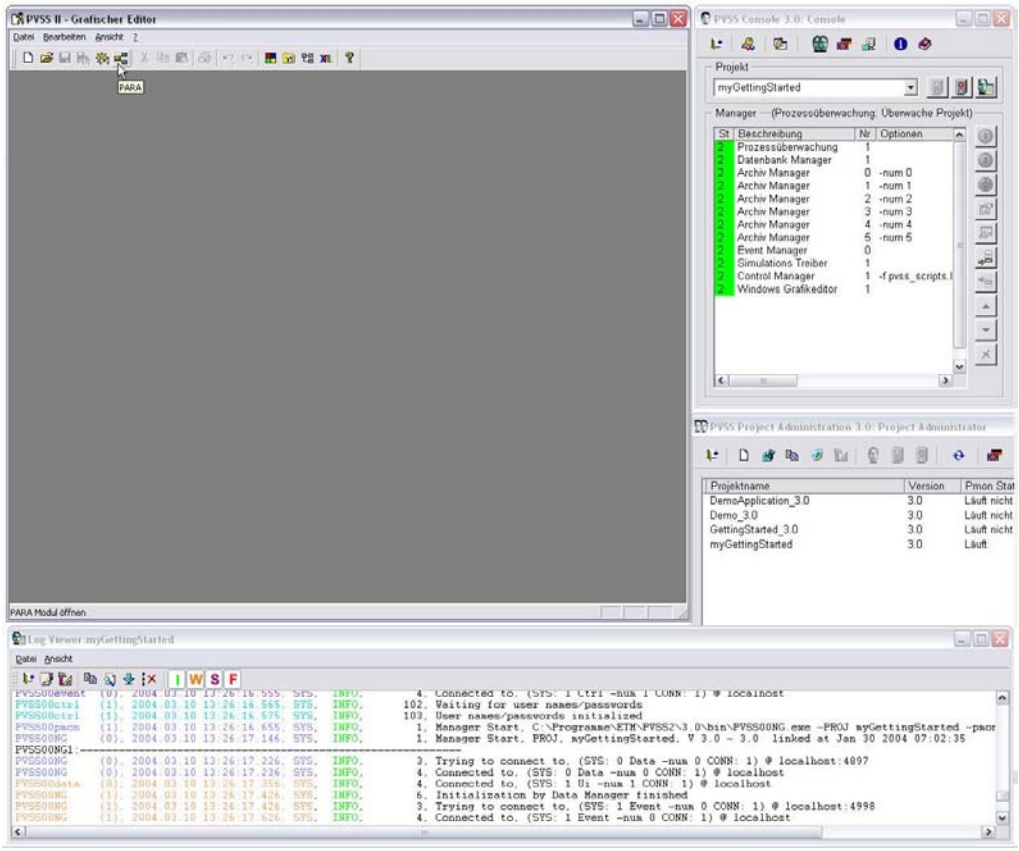


Fig. 4.10
Windows opened after startup: Graphical editor, Console, Projekt Administration, Log Viewer

All managers running correctly are now displayed with the green status colour in the Console. If your project startup was interrupted when the Event Manager was started, and the error message shown in Fig. 3.7 is displayed, then please read the Licensing topic in section 3.5.

The Project Administration program is not needed for the time being and can be closed. The Console is also not relevant to the next steps, but can be left open during the project creation process. In many of the following procedural steps, the Log Viewer **flags up any possible input errors**; it should therefore be placed at the bottom of the screen, for instance, as a sort of **information panel**.

The following first steps into the world of structured datapoints can be made from the user interface of the GEDI graphical editor, which is already open.

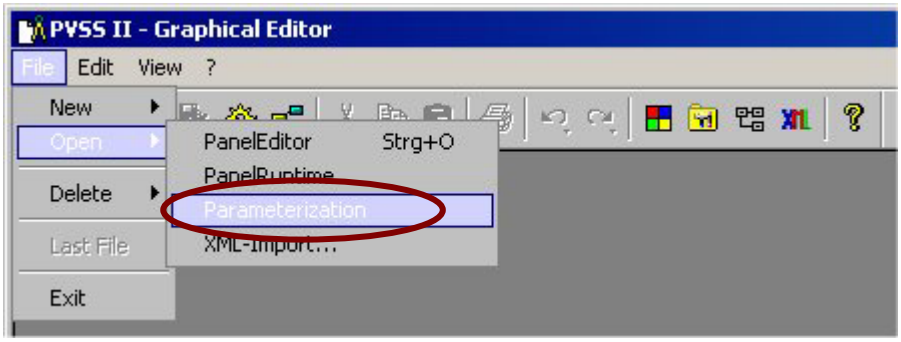


Fig. 4.11
Opening the PARA database editor from the Graphical editor user interface

5 Data model

PVSS II lets you **create** your own **personal device-oriented datapoints**. The basic concept behind structured datapoints has been presented in section 2.5. The following sections explain how to work with datapoints in practice and also provide more in-depth background information.

5.1 Datapoints as information carriers

In order to work with PVSS II and be able to create user interfaces for operating the process, the **carriers of the information to be displayed** must first be defined. These carriers are essentially variables whose values each represent a live item of process information. These **process variables** are called **datapoint elements** in PVSS II.

The PARA **database editor** is the tool normally used to display existing datapoints or create new ones. This can be opened from the GEDI graphical editor already running using the icon.

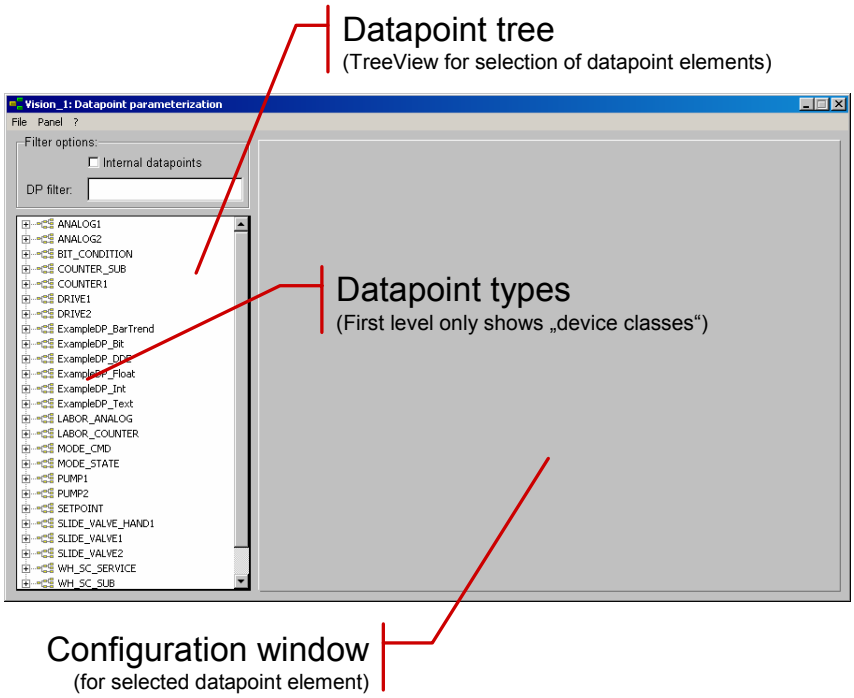


Fig. 5.1
PARA database editor after startup

5.1.1 Datapoint types

After opening the database editor, the list in the left-hand field in Fig. 5.1 displays all **datapoint types** available in the active project. Each of these datapoint types **represents a whole class of devices or logical units**. Clicking on the "+" sign in front of the name of the datapoint type opens the list of **datapoints** (instances/devices) of this type.

A **datapoint type is a sort of template** for structured datapoints. **Structure, name** and sometimes also parameter settings are specified when the type is defined.

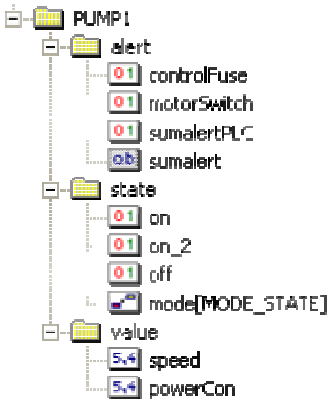


Fig. 5.2
Datapoint type "**PUMP1**" as template for a whole class of devices

Before one can create a structured datapoint as representative of a device, a corresponding datapoint type must be created as a template.

5.1.2 Datapoints

Each **datapoint** can represent a **real device** or a **logical grouping of information**. It is made up of one or more datapoint elements in almost any structure. Each datapoint belongs to its datapoint type, and is consequently displayed with its type in the PARA module. Double-clicking on the name of the datapoint type or the "+" sign in front of the name opens the list of all datapoints of this type.

Fig. 5.3 shows two **types provided as standard** and their datapoints (instances). The datapoint type **ExampleDP_Float** has 4 datapoints (instances) in this case. This datapoint type consists of **just one datapoint element** and can therefore only represent one process variable. This corresponds to the typical data model used in most SCADA systems, but is the exception rather than the rule for PVSS II. This type (and also the other ExampleDP* types) are provided **merely for test and demonstration purposes**.

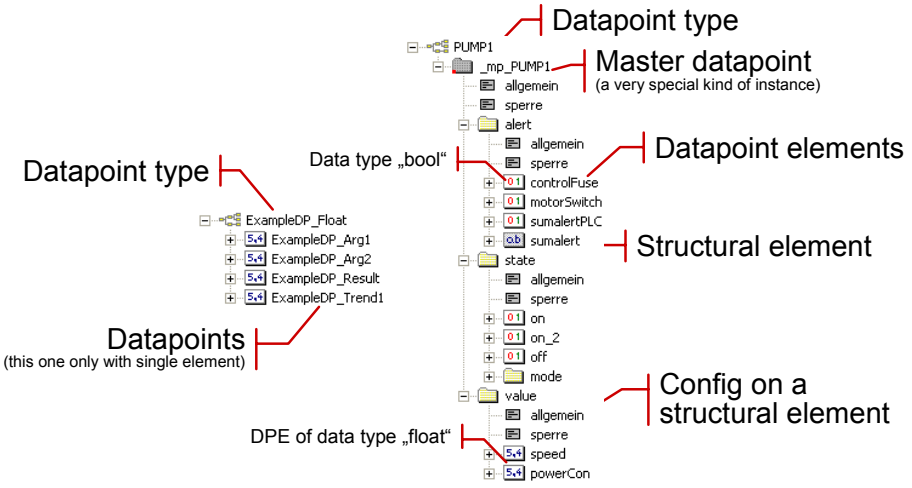


Fig. 5.3
Ready-made instances of the datapoint types **ExampleDP_Float** and **PUMP1**

The **datapoint type PUMP1** shown on the right is more typical of the datapoint structure used in PVSS II. In this case a large number of process variables **make up the device-oriented datapoint**. To start with, **PUMP1** has just a single datapoint instance. Even this is an exception: this is the **master datapoint (MP)**, which can display certain parameterization data for the type. The Online Help contains more details on

this **{Project design ⇨ Mass parameterization ⇨ Datapoint type, Master Datapoint...}**. There are no real datapoint instances yet for the type **`PUMP1`** because we have started with a new, initially empty project.

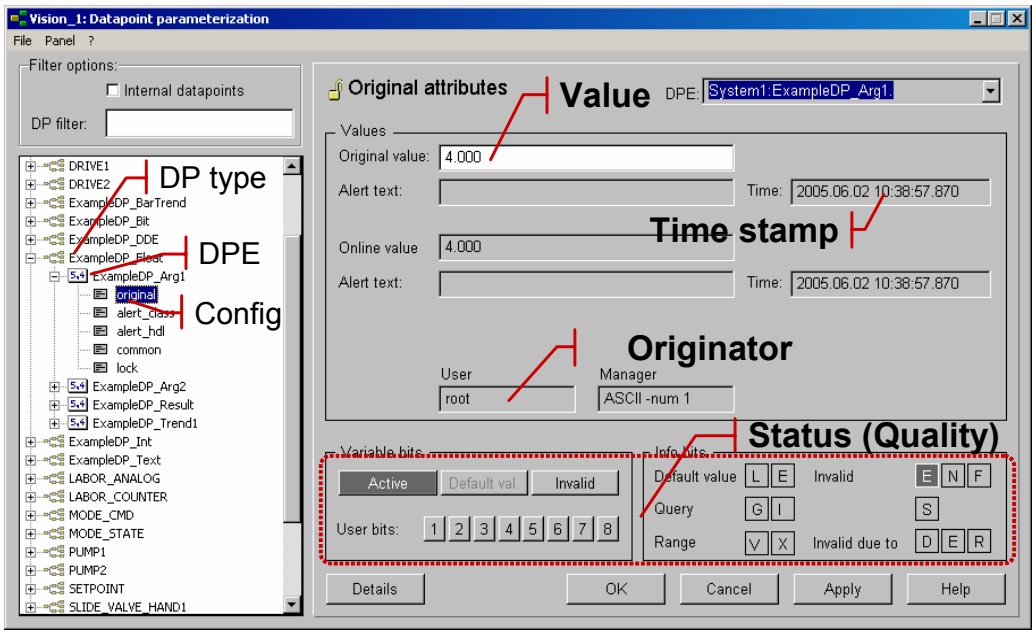


Fig. 5.4
PARA database editor - datapoint type **`ExampleDP_Float`** with datapoint **`ExampleDP_Arg1`**

Datapoint types and datapoints only exist within a project. Each of the projects on a computer can have other datapoint types and datapoints even with the same names. The Import/Export option in the ASCII-Manager can be used to transfer datapoint types or datapoints already created in one project to another project. Online Help **{ASCII Manager ⇨ ASCII Manager Panel}**.

5.1.3 Information at the datapoint element

Each datapoint element represents primarily **the value** of a process value, but there is also some additional information about this value that is very important. In PVSS II one refers to a **complete process image**, because it represents **all the key attributes** of a captured value.

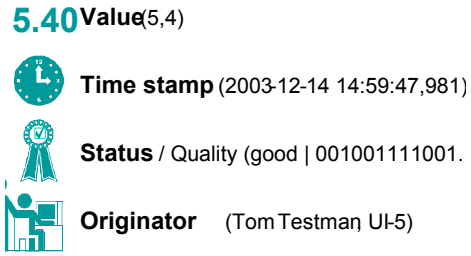


Fig. 5.5
Information at the datapoint element - value with attributes (time, quality, originator)

These additional attributes of the value can also be seen in the PARA database editor (Fig. 5.4). This screenshot also shows that there are two **different fields for the value display**:

Original value	The value that was originally entered i.e. adopted by a controller. This original value is always retained and can still be retrieved after automatic corrections (default value enabled). Active write actions are always performed on the original value.
Online value	The value that all function modules within PVSS II work with. Normally it is an identical copy of the original value. Only if a value has been identified as invalid for instance may a default value be displayed here. When reading a datapoint element (display, processing), it is always the online value that is accessed.

You can find more information on the subject of **original-value preservation** in the Online Help **{PARA module ⇨ Datapoint configs ⇨ _original (Original attributes)}.**

5.2 Modelling datapoint types

In the previous sections we have looked at existing datapoints available from the start of any project. Usually, however, we want to **design our own** device-oriented datapoints, in order to fulfil the **specific project or industry requirements** as closely as possible.

As part of a small **example**, we need a couple of datapoints anyway, so we will configure their types ourselves. A `[right mouse-click]` on the empty, white area of the datapoint tree opens a context menu containing the option "**Create datapoint type**".

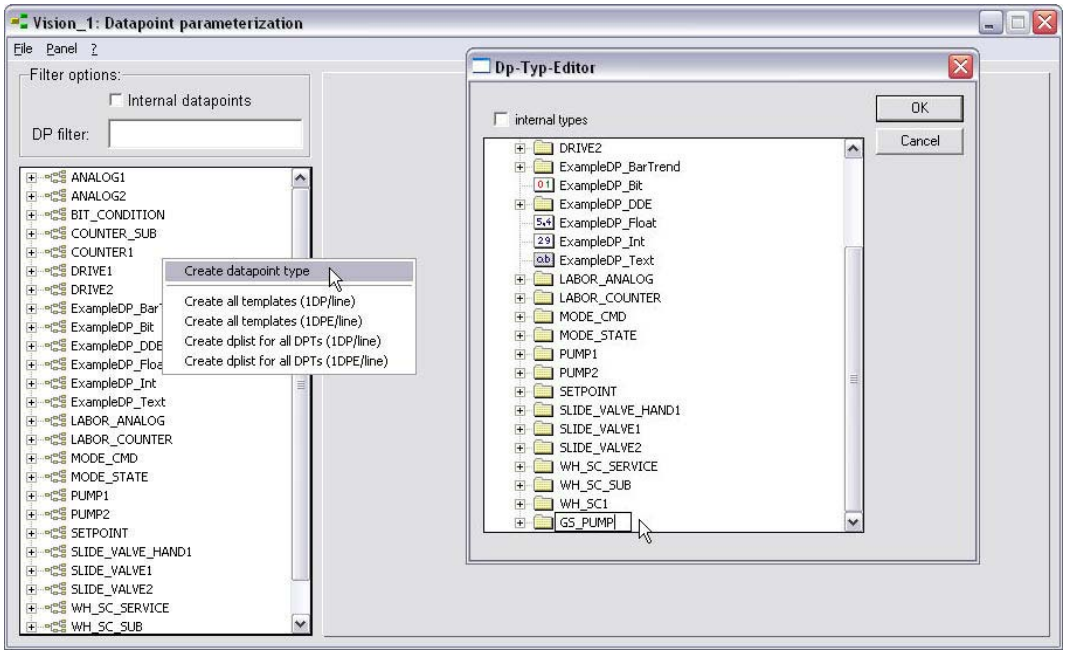


Fig. 5.6
Creating a new datapoint type "**GS_PUMP**" in the Dp-type editor of the PARA module

Selecting this option opens the datapoint type editor in which the required **data structure can be constructed easily**:

1. Replace the type name "NewDpType", already highlighted for renaming, with "**GS_PUMP**".
2. Open the context menu with a `[right mouse-click]` and select "**Insert node**".

3. Assign the name "state" to the inserted node.
4. Repeat the "Insert node" operation directly at the node "state" to insert the sub-node "on".
5. Assign the "**bool**" **data type** to the node "on" by a `[right mouse-click]` ⇨ **Element type**

Structure elements are identified by the folder icon, and **outer elements** have an icon that indicates the specific data type (bool, int, float, etc.). Thus the 01 icon indicating a binary variable is displayed for the "**on**" element in the far right of Fig. 5.7.

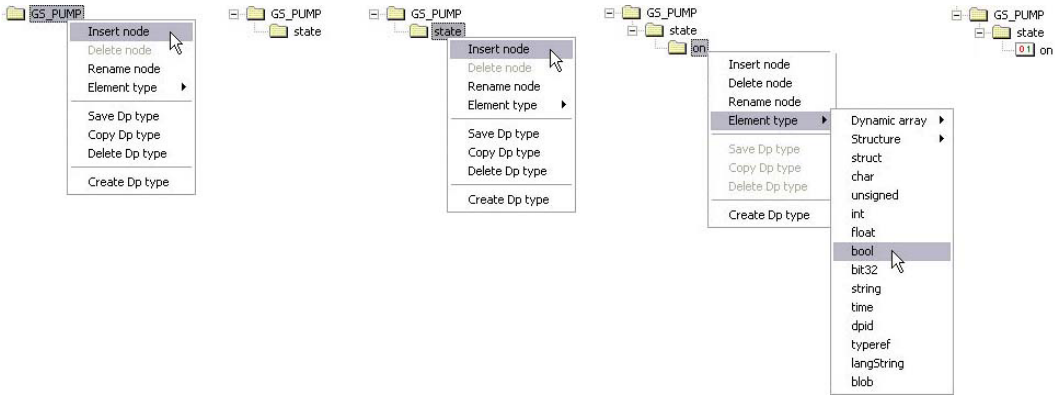


Fig. 5.7
Creating a structured datapoint type in the Dp-type editor of the PARA module

Repeat steps 2. to 5. as many times as is necessary to construct the **complete datapoint type** as shown in Fig. 5.8. Make sure that the elements **`.state.speed`** and **`.cmd.speed`** are each of data type "float".

Of course you can also rename and delete nodes in the process as well.

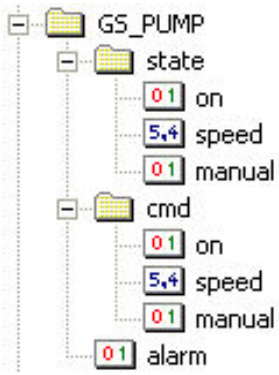


Fig. 5.8
Completed datapoint type **`GS_PUMP`** in the Dp-type editor view

Once you have finished constructing the datapoint type **`GS_PUMP`**, press OK in the Dp-type editor and confirm the prompt with Yes.

5.3 Instancing

Datapoint types alone still **cannot hold any process information** - they are **just a form of template** for the actual datapoints belonging to real devices. In order to **create a device datapoint** from the readymade datapoint type **`GS_PUMP`**, we can again use the context menu from the tree view of the PARA module.

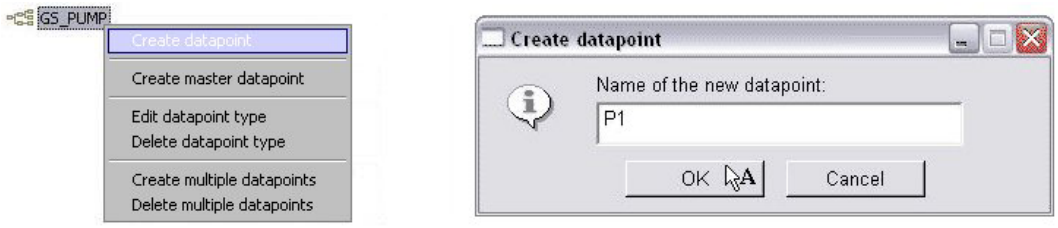


Fig. 5.9
Creating a datapoint "**P1**" of the datapoint type "**GS_PUMP**" in PARA

1. Use the mouse to select the required datapoint type (**GS_PUMP**)
2. `[Right mouse-click]` ⇨ **Create datapoint** opens a dialog box for entering the name
3. Enter "**P1**" for the first pump and confirm with OK

If you now open the tree below **GS_PUMP**, you can see the complete datapoint **P1** of type **GS_PUMP**.

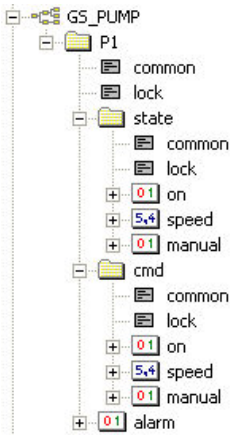


Fig. 5.10
Datapoint **P1** of datapoint type **GS_PUMP** in the structure view of the PARA module

At the moment the elements of the datapoint **P1** should be seen as **purely internal information elements** of the control system, because no **I/O addresses** have been parameterized. Nevertheless, you can still make changes to values, which are detected and processed by all the managers. The simplest way to change the value of an internal datapoint element is in the **Original value dialog box of the PARA module**. This is opened by a `[left mouse-click]` on the element itself or the "original" sub-element (config) in the tree view (see also Fig. 5.4); after entering a new value in the "**Original value**" field and clicking on the Apply button, the new value appears in the "**Online value**" field and the time stamp is updated to give the time at which the new value was applied.

There is **practically no limit to the number of datapoints** that can be created on the basis of the **GS_PUMP** datapoint type. The names of the datapoints must be **unique within the system**. In addition, names must only contain **standard alphabetic characters** (letters), the **digits "0-9"** and the **underscore character "_"**. Every datapoint name must begin with a letter or an underscore.

The datapoint elements are dynamic variables that represent the process or internal states of the control centre. They are managed by the Event Manager and are held in its memory area. Nevertheless, the datapoint elements are variables that can be accessed from any of the managers.

5.4 Settings for the datapoint element

The new datapoint **P1** is now ready for use in our application. It is often desirable, however, to make a few further settings for the elements. For example every element can carry the following **additional information**

- Description
- Alias address (second, internal identification system in addition to the datapoint name)
- Number format (no. of digits before and after decimal point)
- Units

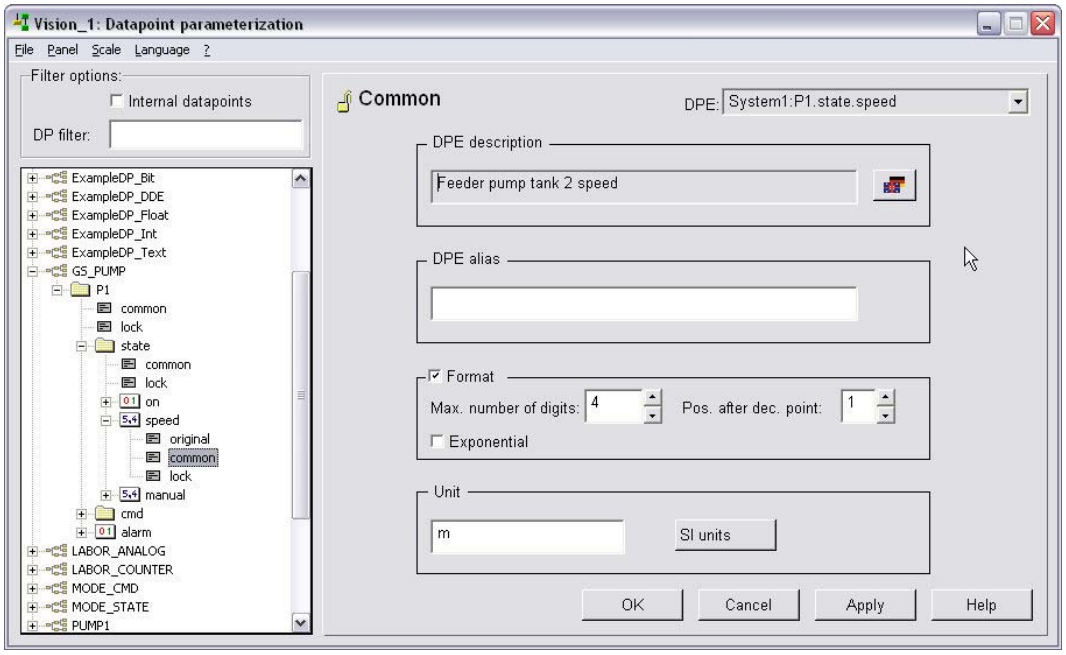


Fig. 5.11
Entering descriptive text, number format and units for the datapoint element (Config: common)

The settings can be displayed by a `[left mouse-click]` on the "common" sub-element (config) for each DPE. Complete the fields for the relevant elements of the datapoint **P1** as shown in Fig. 5.11.

In a multilingual project, you can also enter the description in a choice of languages. The Online Help contains details on multilingual projects **{Multilingual capability}**.

5.5 Addressing

From whatever location in PVSS II, device-based access to an individual element of a datapoint such as **P1** can be achieved by specifying the **full element name**. The **datapoint element name** for the actual speed of pump **P1** is shown in the field "DPE:" in the top-right of **Fig. 5.11**:

```
System1:P1.state.speed
System name:Datapoint name.Element1.Element2. …ElementN
```

The **system name** "System1" exists automatically in every new project created. This system name can also be changed when connecting together several PVSS II systems in a "**distributed system**" (multi-server cluster). We do not need to worry about it further for this example however.

When addressing **local** datapoint elements in the **local system** (which is always the case in this example), the system name can be **omitted**. So we could also address the speed using:

`P1.state.speed`

The Online Help contains more information on addressing datapoints **{CONTROL ⇨ Introduction to CTRL ⇨ Addressing types}**.

5.6 Functions at the device-oriented data object - configs

A number of extra attributes in addition to value, units, description etc., are used to characterize a datapoint element in automation engineering. These include both purely **informative attributes** and attributes used to define **processing** and **alerting methods**.

These additional parameters set **at the datapoint element** are called "**configs**" in PVSS II. We will now demonstrate what the configs mean using the datapoint **`P1`** by way of example: configs shall be used to define a monitored range for the actual or setpoint value of the speed, and to apply a binary Alert handling (alerting) procedure to the element **`.alarm`** .

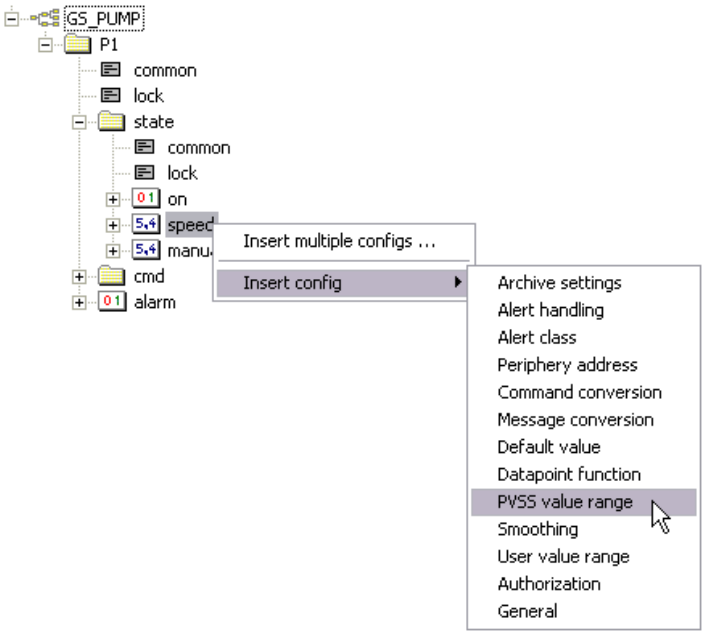


Fig. 5.12
Inserting a range-monitoring config (pv_range) at the DPE **`P1.state.speed`**

To do this, select the element **`P1.state.speed`** with the mouse, open the context menu by a `[right mouse-click]` and select the option **"Insert config" ⇨ PVSS value range.** Open the sub-structure below "**`.speed`**" in the tree view and then click on the "**`pv_range`**" config to **set the range**.

The name of the configs in the tree view in the PARA module may differ depending on the Project engineering language (*paramLang*) that has been set.

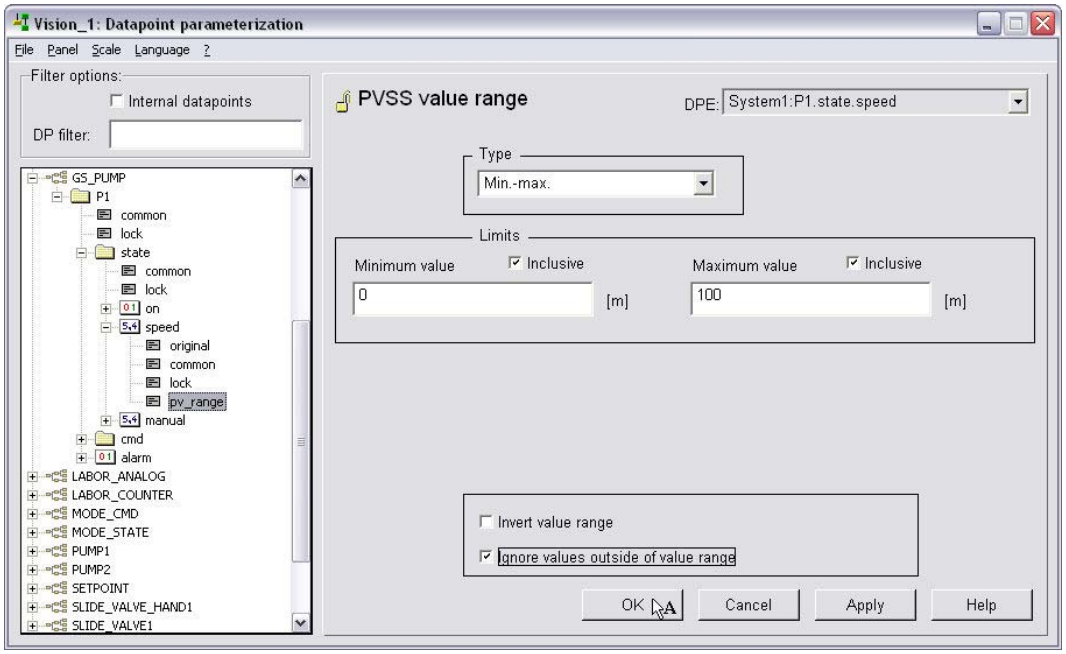


Fig. 5.13
Setting the valid range for **`P1.state.speed`** (config **`pv_range`**)

Define the range as shown above, and then follow the same procedure for the alert handling of the datapoint element **`P1.alarm`**.

1. **Select** the datapoint element **`P1.alarm`** in the tree view of the PARA module.
2. In the context menu `[right mouse-click]` select **"Insert config" ⇨ "Alert handling"**
3. Expand the config structure below **`P1.alarm`** (double-click on the element or click on the "+" in front of the element) and select the config **"`alert_hdl`".**
4. Make the settings shown in Fig. 5.14

    a. Select the **alert class** "060_alert" (alarm with priority 60).
    b. Under **Range texts** enter "Pump fault" for the Alert range (1) and Ok for the Good range.
    c. Select that alert handling shall be performed for the **"`_online._value`**".
    d. **Finally enable** alert handling by selecting the lower-left checkbox.

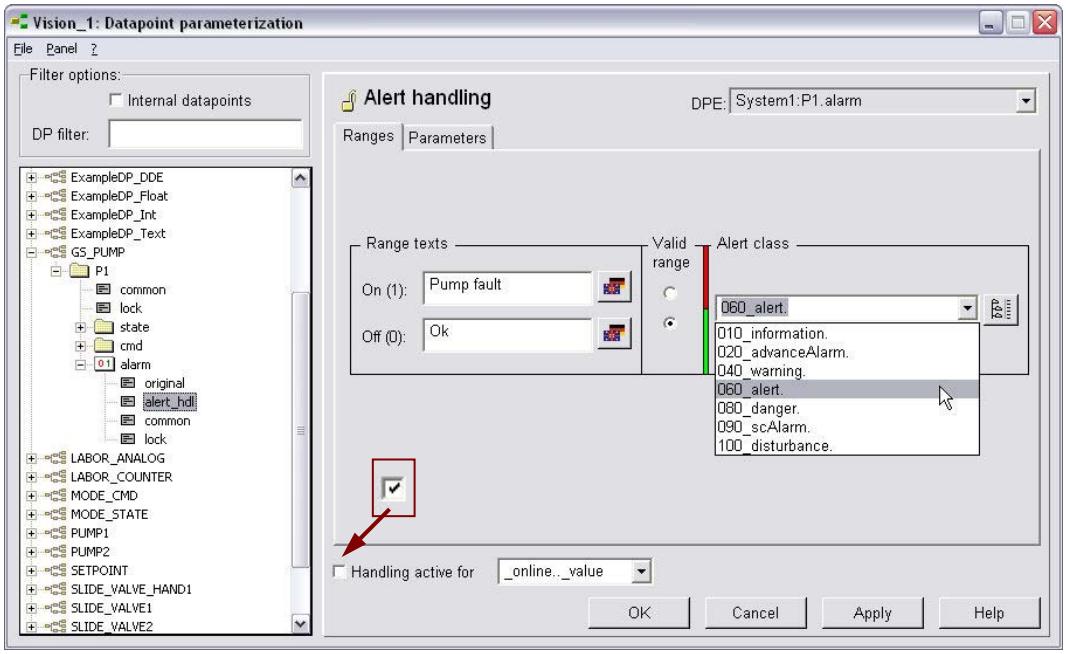


Fig. 5.14
Parameterization of the alert handling (alerting) config for the datapoint element **P1.alarm**

These rules are applied to the datapoint **P1** as soon as the above steps are made - the **parameter settings have been adopted "online"** and can be tested immediately without restarting the system.

Attempts to enter values **outside the enclosed range** [0 to 100] in the original-value dialog box of the speed element **now fail**. If the value for the **P1.alarm** element is set to 1/TRUE, the alert state is displayed in the Alert text field of the original-value view. The alert state colour is also displayed in the Alert handling parameterization window (Fig. 5.14).

Now all the steps required have been performed for direct visualization of the pump **P1** in a process display. Were we working at the level of typical visualization packages, we could proceed with Section 7 on Designing the Process displays. To take full advantage of the **added value offered by object orientation**, however, we advise working through the following sections on creating the data model.

In principle, every datapoint element can have its own **personal set of configs**. Thus the parameterizer defines where a rage-check or an alert-handling is required, thereby configuring his **data model to match his own requirements**. This gives a **lean, scalable memory structure** and saves CPU resources.

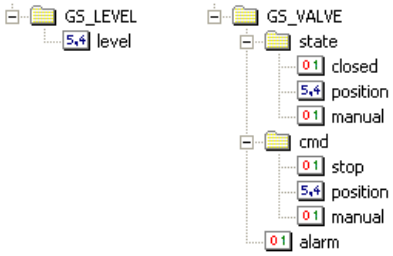
This freedom goes so far as allowing the same elements on datapoints of one and the same datapoint type to accommodate different configs. This certainly pushes the design limits and optimization horizons yet further for the application programmer, but is often not wanted because of the higher engineering time involved and the loss of clarity. This is why PVSS II includes the facility of setting the parameters in common and centrally for all datapoint instances of a type using what is known as a **master datapoint**. See the Online Help **{Project engineering ⇨ Mass parameterization}.**

You can find more information on the configs available and their attributes in the Online Help **{PARA module ⇨ Datapoint configs}.** A comprehensive reference table for all configs and attributes is included in the Online Help **{Reference tables ⇨ Datapoint configs}.**

6 Creating datapoints

6.1 Creating custom datapoints in the PARA database editor

Section 5.3 has already explained with reference to Fig. 5.9 how to create an individual datapoint. This always assumes that a suitably configured datapoint type already exists as a template. In the PARA database editor, it is also possible to create datapoints that "inherit", in addition to the defined structure of their datapoint type, the parameter settings for this type. To do this, however, a "**master datapoint**" must first be created and configured. The following screenshots assume that additional datapoint types (**GS_LEVEL** and **GS_VALVE**) have been created and have a configured master datapoint.



Please refer to the Online Help **{Project engineering ⇨ Mass parameterization}** to find out how to create and configure a master datapoint.

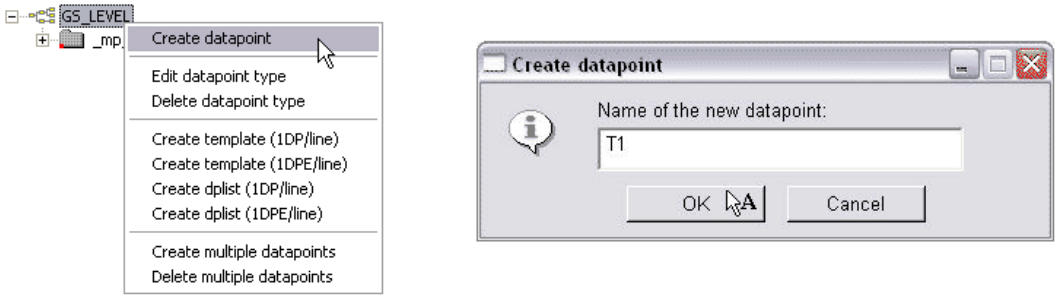


Fig. 6.1
Creating the datapoint **T1** via the context menu of the datapoint type **GS_LEVEL**

To **create** a **single datapoint**, open the context menu for the datapoint type `[right mouse-click]` and click on the "**Create datapoint**" option. Enter the name[1] of the new datapoint in the dialog box that opens; this is created as soon as you click on OK.

1. Create a datapoint **T1** for a datapoint type **GS_LEVEL**
2. Create a another datapoint **T2**

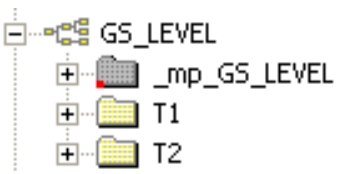


Fig. 6.2
Datapoint type **GS_LEVEL** with master datapoint and the datapoint instances **T1** and **T2**

1 Datapoint names must begin with an alphabetic character (letter) or an underscore "_"; numbers can only be used after this. The case is important - these names are case sensitive.

Datapoints can only be **created** if at least one suitable **datapoint type already exists**. If datapoints are created **without** there being a **configured** master datapoint for this type, then these datapoints only "inherit" the structure, names of the elements and data types.

If there is **also a configured master datapoint**, then the datapoints also receive process-control functions (configs). Which attributes of a config can be set individually at the datapoint is specified by the selection of the respective PowerConfig.

Several datapoint elements at once can also be created in PVSS II in a **single operation** (the instances V1, V2 and V3 of the datapoint type **GS_VALVE** are created below):

1. In the context menu `[right mouse-click]` of the required datapoint type **GS_VALVE**, select the option "**Create multiple datapoints**"
2. In the next dialog box enter "V" in the field "Text before number" as the **prefix for the datapoint name**, and specify the numbering range as "1" to "3" - so a single digit is enough for us in this case (Min. no. of digits = 1).
3. Leave the possible suffix for the datapoint name empty ("Text after number" field)
4. Close the dialog box with OK and confirm the subsequent prompt.

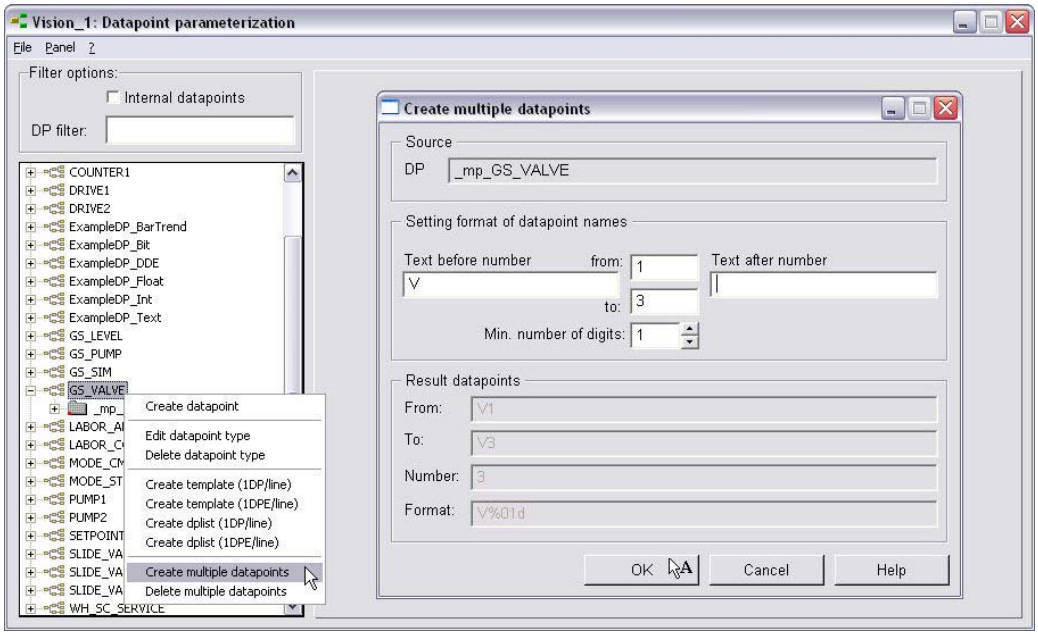


Fig. 6.3
Creating a series of datapoints with automatic naming in the PARA module

As soon as the datapoints have been created, the instance names (e.g. **V1**, **V2** and **V3**) appear under the relevant datapoint type (e.g. **GS_VALVE**) in the tree view of the PARA module.

The PARA database editor is an excellent tool for datapoint design and for prototypes. If, however, large numbers of datapoints (devices) need to be created as the project advances, then this process is normally performed **externally in datapoint lists**. These datapoints are then imported into PVSS II using the **ASCII manager** - the chapters **{Project engineering ⇨ Mass parameterization}** and **{ASCII Manager}** of the Online Help contain more information on this.

In the PARA module, datapoints can also be **deleted** individually via the context menu for the datapoint type or datapoint. Several datapoints at once can also be removed.

7 Graphical user interfaces - "Panels"

7.1 Creating process displays - the graphical editor

In PVSS II, all **graphical user interfaces** for the process operator are called "**panels**". Regardless of whether these are **top-level diagrams**, **process displays**, **detailed views** or **operating** and **information dialog windows**, in PVSS II they are all seen technically as a "panel". They are created using the GEDI graphical editor which we already met in section **4.5**. Switch to the graphical editor or start it via the Console (`[right mouse-click] from the` Option "**Windows Graphical editor**" and click on the option "**Start manager**").

Once open, click on the "**New panel**" icon in the toolbar. Some of the user-interface settings can only be configured in practice with a panel open. Toolbars and other views within the GEDI are implemented as **dockable views** - you can position these elements around the outer edges of the application window to suit, or move them wherever you want as windows.

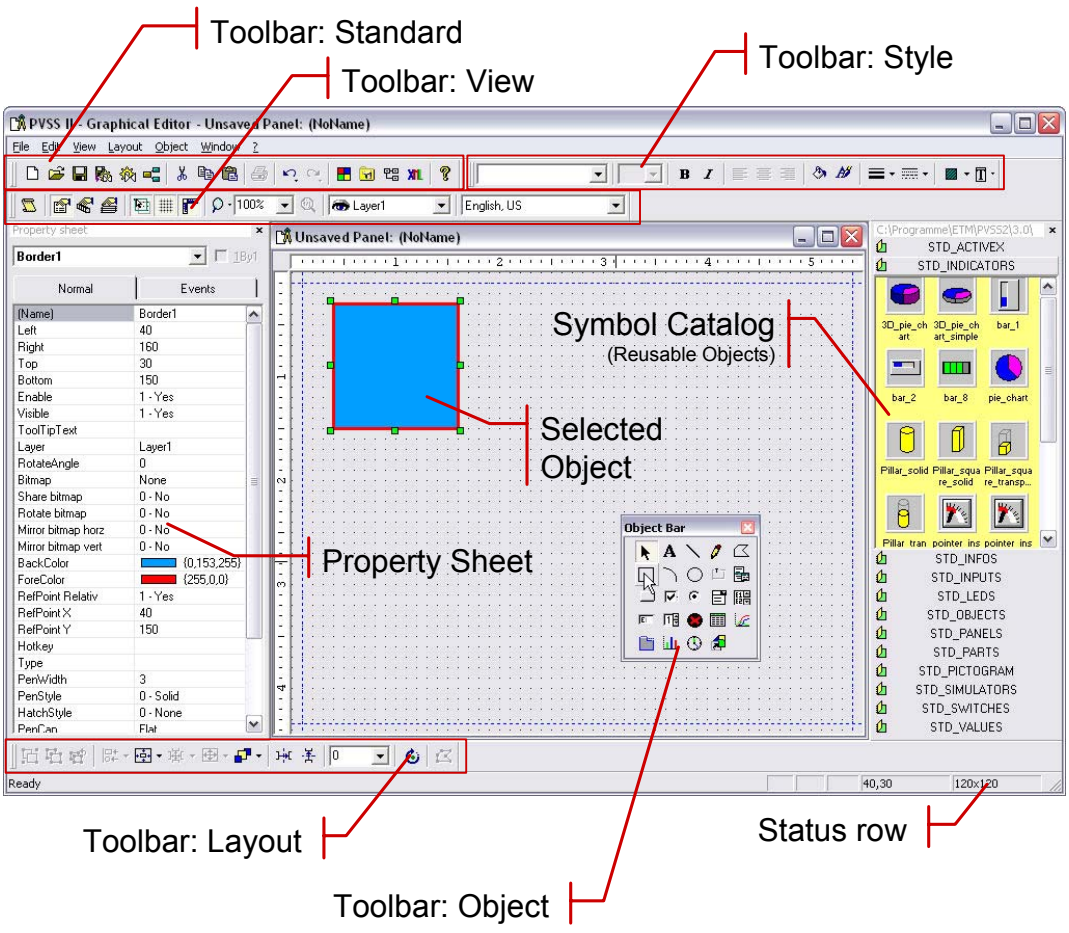


Fig. 7.1
Elements of the GEDI graphical editor: drawing plane, catalog, Property sheet, etc.

The Online Help **{GEDI module ⇨ Native GEDI}** contains a detailed description of the menus and toolbars of the GEDI graphical editor.

You can show and hide toolbars to give the view you want via the menu **View ⇨ Toolbars**. The most important toolbar in the early stages of creating a panel is the **Object** bar.

7.2 Simple drawing operations

A **drawing object is inserted** by selecting a **graphics object** in the **object bar** and **dragging** a line or an 2D shape across the panel surface. Fig. 7.2 names the available **graphics primitives** and **widgets** (complex graphics objects) available for creating user interfaces.

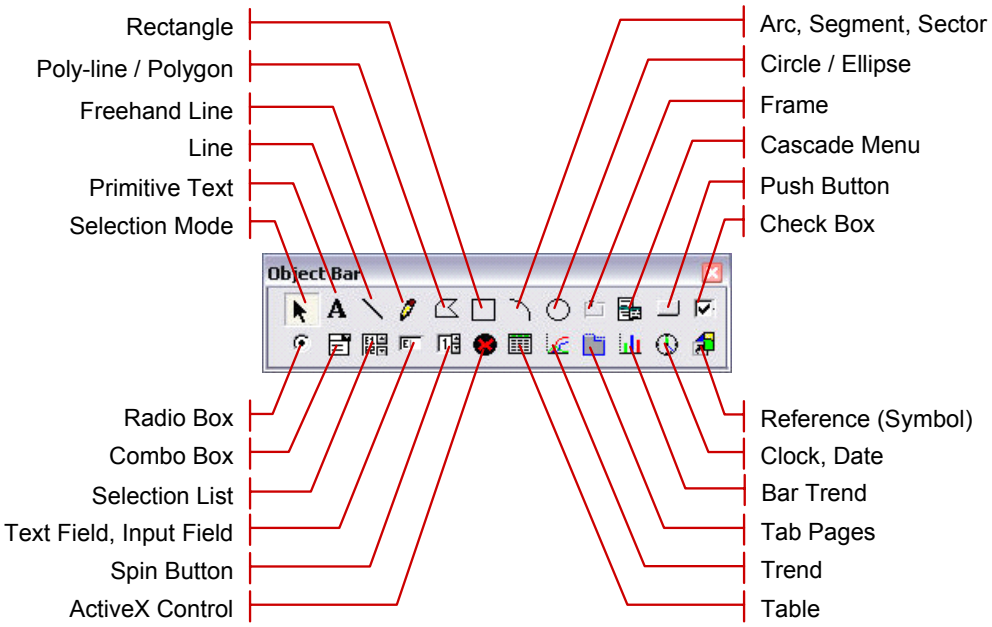


Fig. 7.2
Object bar in the Native GEDI: basic elements of the user-interface design

Once the new graphics object has been generated, it initially remains selected for further processing. **To select** a different **object**, simply click `[left mouse-click]` on the object required. The graphics object currently selected is identified by **green, rectangular selection elements** around its outline.

To **change the size** of an object, **drag** the selection elements with the mouse in the required direction. If the mouse cursor is moved over the inner area of a selected graphics object, then the editing mode changes automatically to **move mode**, and the mouse cursor becomes a 4-way arrow; as long as this arrow is displayed, the graphics object can be **moved by dragging** it across the panel surface. (For shapes with a transparent background, drag on the outline.)

To select more than one object at once, hold down the `[Shift]` key after selecting the first object, and click on the additional objects `[left mouse-click]`.

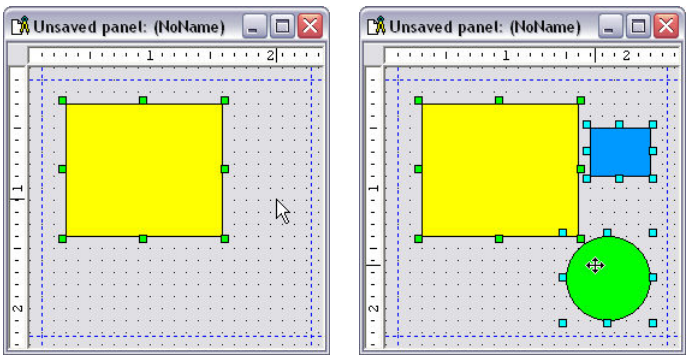


Fig. 7.3
Selected rectangle (left) and multiple selection using the `[Shift]` key (right)

During selection and drawing, the **object position** and the current **object size** are displayed in the **status bar** in the lower right of the GEDI window.

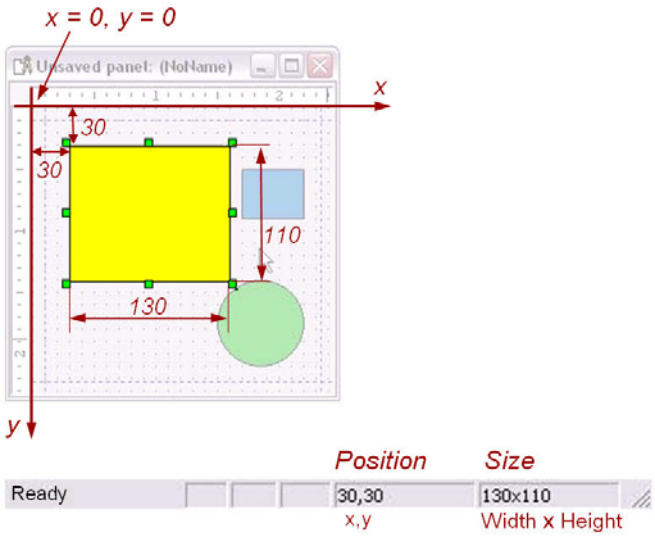


Fig. 7.4
Coordinate system in the panel, readings in the status bar of the GEDI

The **coordinate system** used is based on the Window standard: the origin (0,0) is in the top left corner of the panel; positive x-coordinates increase from left to right, positive y-coordinates from top to bottom. Positive **angles** are in the **mathematically positive** direction (counter clockwise).

For practice, draw a few graphics objects in a new panel and try to change their size, position, colour and orientation etc.

1. **Draw** a 2D shape using the rectangle element (object bar)
2. Select the shape with the mouse and **move** it
3. Alternatively use the arrow keys [←], [↑], [→], [↓] on your **keyboard** to **move** a selected object. The object moves by one pixel every time you press the key, and by a grid division if you hold down the [Ctrl] key at the same time.
4. Click with the mouse on the green selection elements and drag in the directions shown - the **object size** changes
5. With an object selected, click on the tool in the "Format" toolbar to select a different **background color** (fill color, *backcolor*) from the colour selector.
6. With an object selected, click on the tool to select a different **foreground color** (colour for boundaries and lines, *forecolor*) from the color selector.
7. With an object selected, click on the or tool to select a different line thickness or **style** respectively.
8. Try out similar operations yourself and combine them in any way you wish
9. You can now also display the result of your work in the **Runtime view**. You can click on the icon or use the menu option **File ⇨ Save & Preview** to open **a preview** of what it will look like in the runtime user interface. Save the as yet unsaved panel as "*test1.pnl*" in the default path **...\panels** (see also section 7.7)

When creating an ellipse or a rectangle, hold the `[Shift]` key down while dragging with the mouse to obtain an object with **equal dimensions in the x- and y-directions** i.e. a circle or a square. Similarly, hold down the `[Shift]` key to retain **proportionality** when changing the size with the mouse.

If you select an element and then press the arrow keys ←, ↑, →, ↓ with the `[Shift]` key pressed, you will **change** the **size** of the object pixel by pixel.

When drawing a polygon or a polyline, keep holding the **mouse button** pressed **as far as the second point** of the line, and then click for every additional point. You can change the geometry of ready-drawn polygons and polylines via the menu **Layout ⇨ Contour ⇨ Edit points**.

The Online Help **{GEDI module ⇨ Native GEDI}** contains further details on the graphical editor.

7.3 The Property sheet

The **Property sheet** can be opened via the menu **Edit ⇨ Properties** or a `[right mouse-click]` ⇨ **Properties** or using `[Ctrl]+[P]` on a panel / selected graphics object. It displays two tabs containing the main information on an object.

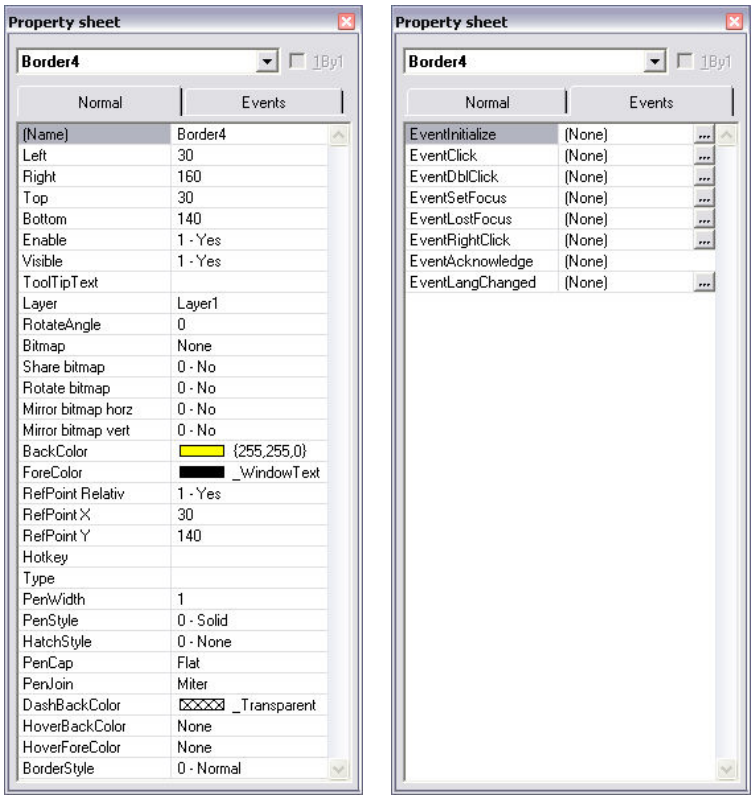


Fig. 7.5
Property sheet for parameterization of graphics objects

The Property sheet can be used both for **displaying information** and for **direct data entry**. All the actions performed previously in section 7.2 with the mouse or via the toolbars **could also be achieved via the Property sheet**.

1. The properties **Left, Right, Top** and **Bottom** can be used to change the position on the drawing plane and to specify a new object size.
2. The **BackColor** and **ForeColor** entries in the Property sheet let you change the **fill color** and **line color** respectively
3. The **line thickness** can be set using **PenWidth** in the Property sheet.
4. The **line style** can be set using **PenStyle** in the Property sheet.

Thus in many respects, the Property sheet constitutes a **second, alternative input option** for designing graphics objects. If the **drawing plane itself** is selected, then the setting options for the panel are displayed in the Property sheet. For example, BackColor can be used to specify the background colour, or the **Width** and **Height** properties used to set the **width and height of the panel** in pixels.

Many of the properties that can be set **in the Property sheet at the parameterization stage**, can also be changed **dynamically at runtime**. This change can occur, for example, as a function of a process variable, i.e. depending on the value of a datapoint element.

The **parameterization stage** means all operations that are performed **during creation** of an application. "At **runtime**" refers to the displays and procedures that take place **during operation**. With the graphical user interfaces, the graphical editor is mainly used at the parameterization stage, and the Native Vision module at runtime.

The "Events" tab, the second tab in the Property sheet, is used for **specifying the dynamic properties of the object that will apply later at runtime**. In PVSS II, all dynamic properties can be seen **as responses to events**. These events could be:

- **Opening** the panel (in which the graphics object is located)
- **Clicking** on the graphics object
- Confirming an **input** by pressing the Enter key
- Receiving or losing the **input focus**
- …

Whenever such an event occurs, **an action is executed automatically**. The operations that make up such an action after an event need to be defined. This can be done either via a Wizard (see section 7.4) or by **programming a** Script (see 7.5).

If you wish to display an **automatically updated process value** in a panel, then this instruction must be specified for the event "Opening the panel" i.e. in **EventInitialize**. Then right at initialisation, the panel registers (connects) itself to all changes in value of the datapoint element. Every subsequent change in value then automatically results in execution of the "response", in this case the correct display of the value.

7.4 Making graphics properties dynamic (Simple Parameterization)

In order to be able to change **properties of a graphics object as a function of process variables** (datapoint elements), an appropriate instruction is parameterized in the GEDI graphical editor using a **Wizard** ("Simple Parameterization"). The following section shows how to implement a **status indicator** in the form of a coloured circle symbol for the Pump P1 data model created earlier.

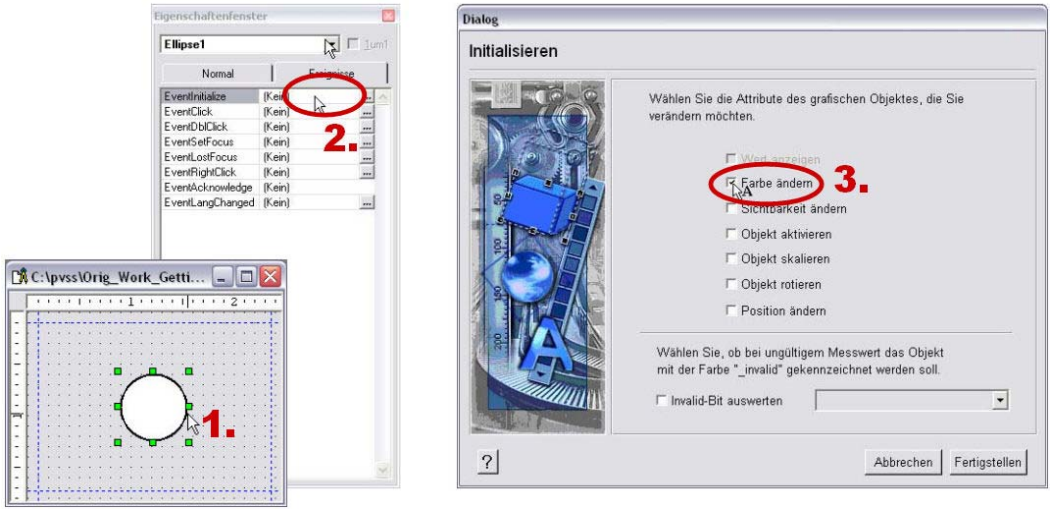


Fig. 7.6
Dynamic changes to graphics properties - parameterization in the graphical editor

1. Use the mouse to **select** the graphics object concerned
2. In the "**Events**" tab of the **Property sheet**, click on the white column area beside "**EventInitialize**". (You need to select this event because the operating-status indicator for **`P1`** needs to be displayed as soon as the process display is opened, and then continuously updated)
3. A **Parameter dialog box** opens; click on the "**Change color**" checkbox and click on Next
4. In the next step of the Wizard, **select** the "**background color** (fill color, text background") checkbox in the top section and enable the "**dependent on value**" option in the lower section. Click on Next to switch to the last step of the wizard.
5. Click on the "datapoint selector" button  on the left of the "datapoint element" input field
6. The **Datapoint selector** opens; navigate through the tree view to the element **GS_PUMP ⇨ P1 ⇨ state ⇨ on** and click on OK.
7. The selected datapoint element **`P1.state.on`** appears in the **Datapoint element input field**. (You can also enter this character string directly)
8. Using the combo-boxes, select a preset colour for the **value 0 (FALSE)** and the **value 1 (TRUE)** respectively. Alternatively you can select any colours graphically using the **Color selector**.
9. Click on Finish to conclude the parameterization process.

Fig. 7.7
Steps in the Wizard for configuring an animation for value-dependent display of the background color

Test the finished result using the graphical editor Preview function (see section 7.7). The easiest way **to change** the value of the datapoint element is to enter the value (0 / 1 or TRUE / FALSE) in the "Original value" field in the Original value dialog window of the PARA module (see **Fig. 5.4**) and click on Apply to set it.

You can follow the same procedure to parameterize value displays, bar graphs, buttons for issuing commands or changing screens and many other parameterization options. Online Help **{GEDI module ⇨ Native GEDI ⇨ Simple Parameterization}.**

7.5 Scripting in the graphic

As an alternative to using the Wizard, a **user-defined script** can also specify the dynamic properties of graphics objects. For this option, **do not** click on the white column area in the Events tab of the Property sheet but **on the button with the three dots** "..." on the far right; this opens the **Script editor**, where you can create almost any animation instructions you wish.

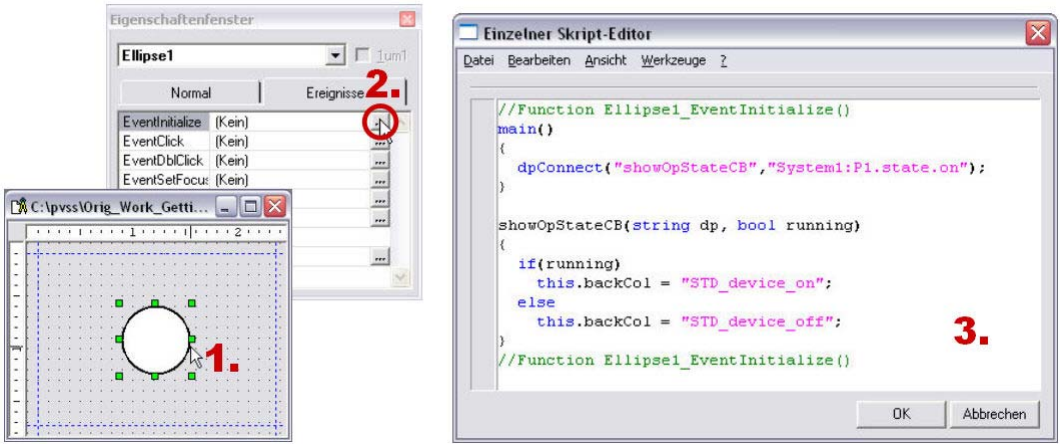


Fig. 7.8
Creating a custom animation by writing a Control script

The script can contain programming code to **access current and historical data** as well as numerous **graphics properties**. Although the Control language, with a syntax based on ANSI-C, is easy to learn, it falls far outside the bounds of this document. You can find out more about it in the Online Help **{CONTROL ⇨ Introduction to CTRL** or **CONTROL ⇨ Control graphics}.**

7.6 Using ready-made symbols for displaying data

The simplest way to create a graphical user interface is **to use off-the-shelf symbols** (references). PVSS II comes with a whole set of these objects. They are a useful aid when getting started, and can also be used as examples or building blocks in symbols designed yourself.

Open the **Catalog view** containing the ready-supplied graphics objects by selecting the menu option **View ⇨ Catalog ⇨** *C:\Program files\ETM\PVSS2\3.0* (if you have selected a different program directory, specify the path accordingly)

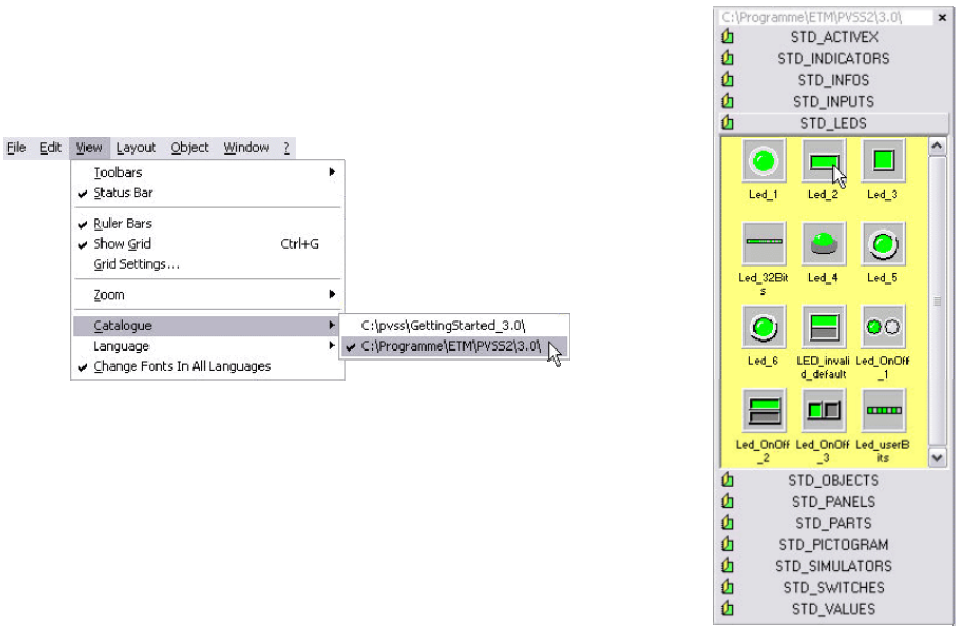


Fig. 7.9
Dsiplaying the ready-made symbols via the View menu in the graphical editor

10. Create a **new panel** and save it under the name "*myFirstPanel.pnl*"
11. In the catalog section "STD_LEDS", select the **standard symbol** "Led_2" and drag it into the top left of the drawing plane.
12. The **parameterization dialog box** opens; click on the "datapoint selector" button  on the right of the "datapoint" input field
13. The **Datapoint selector** opens; navigate through the tree view to the element **GS_PUMP ⇨ P1 ⇨ state ⇨ on** and click on OK.
14. The selected datapoint element **`P1.state.on`** appears in the **Datapoint input field**.
15. **Remove** the check mark from the "Alert handling color" checkbox
16. Click on OK in the parameterization dialog box for Led_2
17. Using the mouse, drag the inserted symbol Led_2 to make it slightly bigger.

The panel just created is now able to display the **'ON' operating status of our pump `P1`** in the form of a stylised LED indicator. We will also provide a suitable input device so that we can **change this value directly from the panel as well**. In addition, we want to display the current **actual speed value as a pillar indicator** and make it possible **to enter** this analogue value in a **text field**.

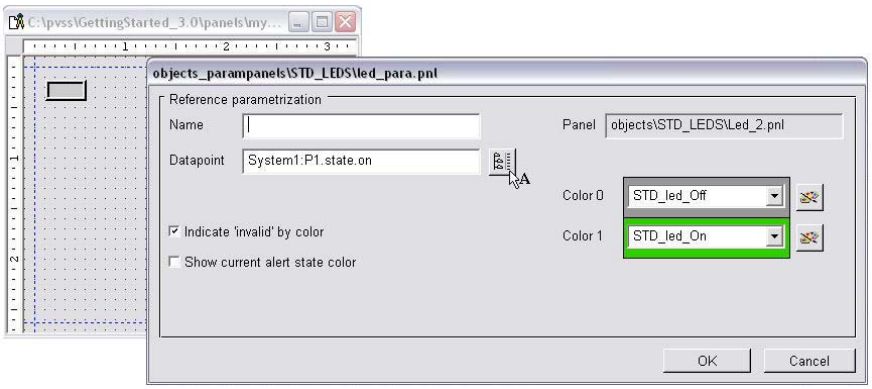


Fig. 7.10
Inserted standard symbol **Led_2** with parameterization dialog box for specifying the datapoint element

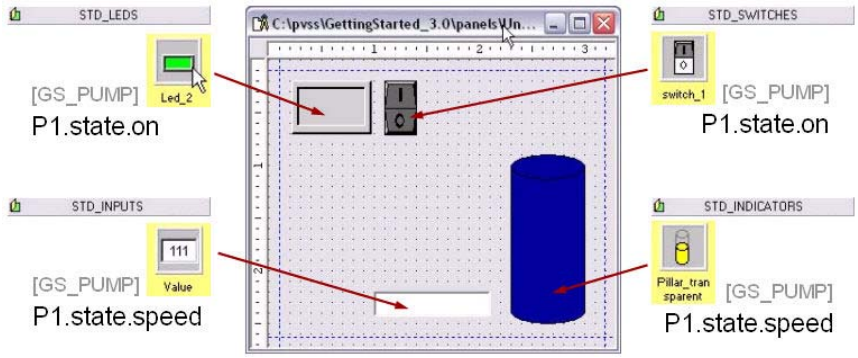


Fig. 7.11
Using ready-made standard symbols for creating panels

1. Drag the **standard symbol "switch_1"** from the "**STD_SWITCHES**" catalog section into the panel and place it beside the LED indicator.
2. As before, select **`P1.state.on`** in the "**Datapoint**" field of the parameterization box.
3. Drag the **standard symbol "Pillar_transparent"** from the "**STD_INDICATORS**" catalog section into the bottom right of the panel.
4. Specify the element **`P1.state.speed`** for the datapoint in the Parameterization dialog box.
5. Also insert the **standard symbol "Value"** from the "**STD_INPUTS**" catalog section in the panel, placing it beside the pillar indicator.
6. Specify again the element **`P1.state.speed`** for the datapoint in the Parameterization dialog box.

In all four standard symbols used so far, the "Datapoint" input field of the parameterization dialog box initially contained the text "**`$dpe_value`**". This is what is known as a **$-parameter** (say "dollar parameter"), which acts as a placeholder. For now we just need to accept that the first part tells us the data type of the expected input, and the second part the type of the information sought. You can find more information on this in the Online Help **{CONTROL ⇨ Introduction to CTRL** or **CONTROL⇨ Introduction ⇨ CTRL parameters}.**

`$dpe_value`	⇨	**`dpe`**...	Datapoint element
		`value`...	Value

This is why you had to enter the **name of the datapoint element** whose actual value was to be displayed via the standard symbol.

7.7 The Preview in the graphical editor

Click on the icon or select the menu option **File ⇨ Save & Preview** to display the panel you have just created in the Runtime preview of the GEDI. Click on the stylised switch and watch how the colour of the LED indicator changes.

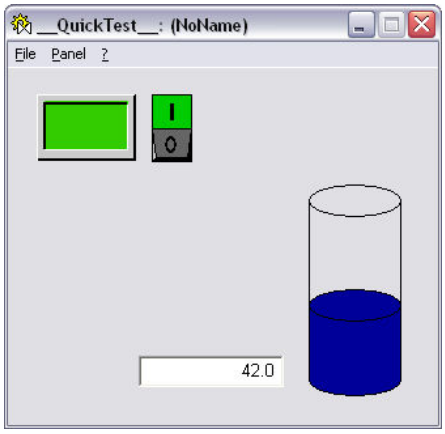


Fig. 7.12
Runtime preview of the panel "**myFirstPanel.pnl**" in the graphical editor (Save & Preview)

Click in the input field and type in a numerical value from the keyboard. The fill level of the pillar symbol changes accordingly.

Alternatively, you can test that your panel display is working by entering the appropriate value in the PARA Original value dialog box. To do this, switch to the PARA module and navigate to the datapoint type **GS_PUMP**. Then navigate to the element **P1.state.speed** and select the **original** sub-node. In the dialog box that opens (similar to Fig. 5.4), enter a new value in the "Original value" field and then click on Apply; the new setting is then displayed in the graphic.

The numerical value entered for **P1.state.speed** must lie inside the enclosed range [0 to 100] because we have specified this in Fig. 5.13. If we attempt to set a value outside the range we are notified immediately in a dialog box that opens from the selected standard symbol, and the value is not adopted.

The Preview (Save & Preview) of the GEDI graphical editor matches as closely as possible the display in the Native Vision runtime user interface. Nevertheless, note that a few procedures may run more slowly in the preview (CTRL scripts, Hover colours, etc.). Panel-in-panel displays are only shown properly in Native Vision.

When being used for normal operation of a process, the operating panels are displayed in their own runtime user interface (Native VISION). This is added to the Console using the entry **PVSS00NV...Windows User interface**. A required process display can be opened automatically immediately at start up using the parameter **-p *panelname.pnl***.

8 Contact

8.1 Head office

ETM International / Austria

ETM professional control GmbH
Kasernenstraße 29, A-7000 Eisenstadt
Phone: +43-2682-741-0 Fax: +43-2682-741-107
E-Mail: info@etm.at
Web: www.etm.at, www.pvss.com

8.2 Sales

Phone: +43-2682-741-0, +43-2682-741-144

Fax: +43-2682-741-107
E-Mail: sales@pvss.com

8.3 Licensing

Phone: +43-2682-741-0 Fax: +43-2682-741-107
E-Mail: license_pvss@etm.at

8.4 Training

Phone: +43-2682-741-0 Fax: +43-2682-741-107
E-Mail: competence@etm.at

8.5 Support/Engineering

Kasernenstraße 29, A-7000 Eisenstadt
Phone: +43-2682-741-0 Fax: +43-2682-741-107
E-Mail: product_center@etm.at