

PVSS Introduction for Newcomers

1 Introduction

PVSS II is a SCADA system. SCADA stands for Supervisory Control And Data Acquisition. PVSS will be used to connect to hardware (or software) devices, acquire the data they produce and use it for their supervision, i.e. to monitor their behaviour and to initialize, configure and operate them. In order to do this PVSS provides the following main components and tools:

- **A run time database**
the place where the data coming from the devices is stored, and can be accessed for processing, visualization, etc. purposes.
- **Archiving**
Data in the run-time database can be archived for long term storage, and retrieved later by user interfaces or other processes.
- **Alarm Generation & Handling**
Alarms can be generated by defining conditions applying to new data arriving in PVSS. The alarms are stored in an alarm database and can be selectively displayed by an Alarm display. Alarms can also be filtered summarized, etc.
- **A Graphical Editor (GEDI/NG)**
Allowing users to design and implement their own user interfaces (panels).
- **A Scripting Language**
Allows users to interact with the data stored in the database, either from a user interface or from a “background” process. PVSS scripts are called **CTRL** (read control) scripts and follow the C syntax and include many SCADA-specific functions.
- **A Graphical Parameterization tool (PARA)**
Allowing users to:
 - Define the structure of the database
 - Define which data should be archived
 - Define which data, if any, coming from a device should generate alarms
 - etc.
- **Drivers**
Providing the connection between PVSS and hardware or software devices to be supervised. Common drivers that are provided with PVSS are OPC, ProfiBus, CanBus, Modbus TCP/IP and Applicom. A DIM driver is provided as part of the Framework.

2 Architecture

PVSS has a highly distributed architecture. A PVSS application is composed of several processes, in PVSS nomenclature: **Managers**. These Managers communicate via a PVSS-specific protocol over TCP/IP. Managers subscribe to data and this is then only sent on change by the Event Manager, which is the heart of the system (see below).

Furthermore, Drivers can be configured to only send data to the Event Manager when a significant change is seen. Hence, in a correctly configured system there is essentially no data traffic in stable conditions, i.e. when the process variables are not changing.

As can be seen in Figure 1 below a typical PVSS application is composed of a number of Managers¹:

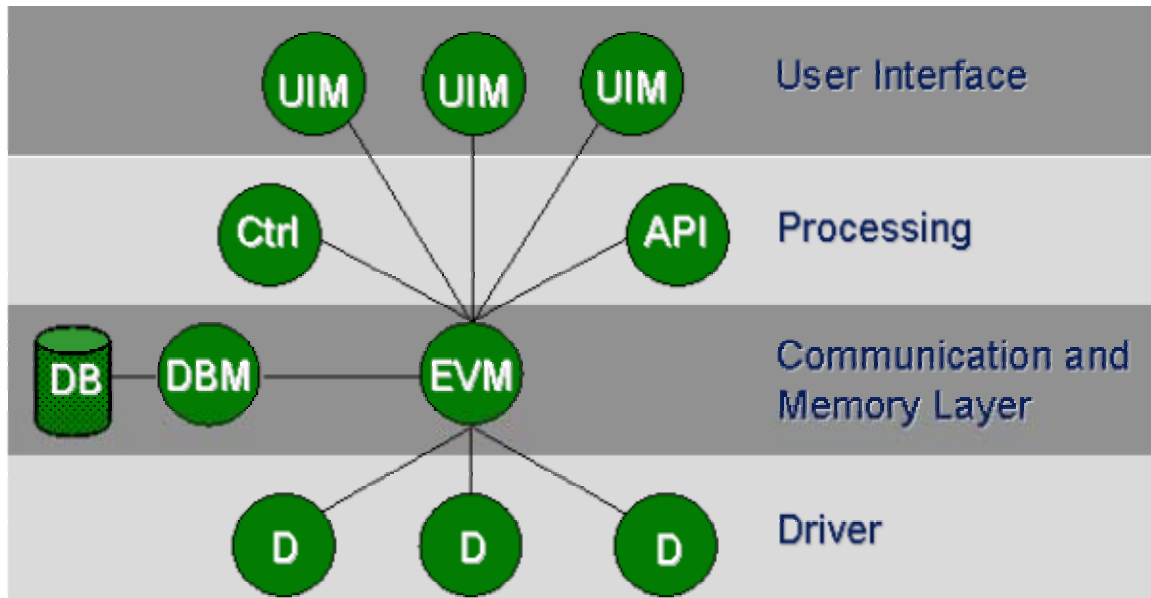


Figure 1: A Typical PVSS System

The Event Manager (EVM) – is responsible for all communications. It receives data from Drivers (D) and sends it to the Database Manager to be stored in the data base. However, it maintains the “process image” in memory, i.e. the current value of all the data. It also ensures the distribution of data to all Managers which have subscribed to this data.

The DataBase Manager (DBM) – provides the interface to the (run-time) data base.

User Interface Managers (UIM) – can get device data from the database, or send data to the database to be sent to the devices, they can also request to keep an “open” connection to the database and be informed (for example to update the screen) when new data arrives from a device. There is a UI for Linux and a Native Vision (NV) for Windows. The UIM can also be run in a development mode; PARA for parameterization of DPTs/DPs, GEDI for the Graphical Editor (GEDI for Linux and NG for Windows).

Ctrl Managers (Ctrl) – provide for any data processing as “background” processes, by running a scripting language. This language is like “C” with extensions.

¹ Please note that not all possible Managers are shown in the figure.

API Managers (API) – Allow users to write their own programs in C++ using a PVSS API (Application Programming Interface) to access the data in the database.

Drivers (D) – Provide the interface to the devices to be controlled. These can be PVSS provided drivers like Profibus, OPC, etc. these will be described later in more detail, or user-made drivers.

Archive Managers – Allows users to archive data for later retrieval and viewing. A project, which is the name for a PVSS application, may have one or more Archive Managers and one can configure which data is stored in which manager.

ASCII Manager (ASCII) - Allows users to export/import the configuration of a project (DPTs/DPs – see later) to/from an ASCII file. It allows users to select which data should be exported. The resulting files can be modified, e.g. using Excel, and re-imported. The import feature can be very useful for transferring DPTs/DPs from one project to another and is used when add Framework components to a project.

All PVSS applications, as well as the majority of PVSS tools, are essentially built of panels and scripts. Panels can be created using the Graphical Editor and may include static or dynamic widgets. Dynamic widgets require CTRL scripting to create the required behaviour. Typical behaviour would be the setting of values to the hardware or the display of data originating from the hardware.

A **PVSS System** is an application containing one data base manager and one event manager and any number of drivers, user interfaces, etc.

PVSS Managers can run on Windows or Linux and they can all run in the same machine or be distributed across different machines (including mixed Windows and Linux environments). When the managers of one system run distributed across different machines this is called a **PVSS Scattered System**.

PVSS can provide for very large applications, in which case one PVSS system would not be enough. In this case a **PVSS Distributed System**, a confederation of communicating PVSS systems, can be used. As shown in Figure 2 a distributed system is built by adding a Distribution Manager (Dist) to each system and connecting them together. Hundreds of systems can be connected in this way.

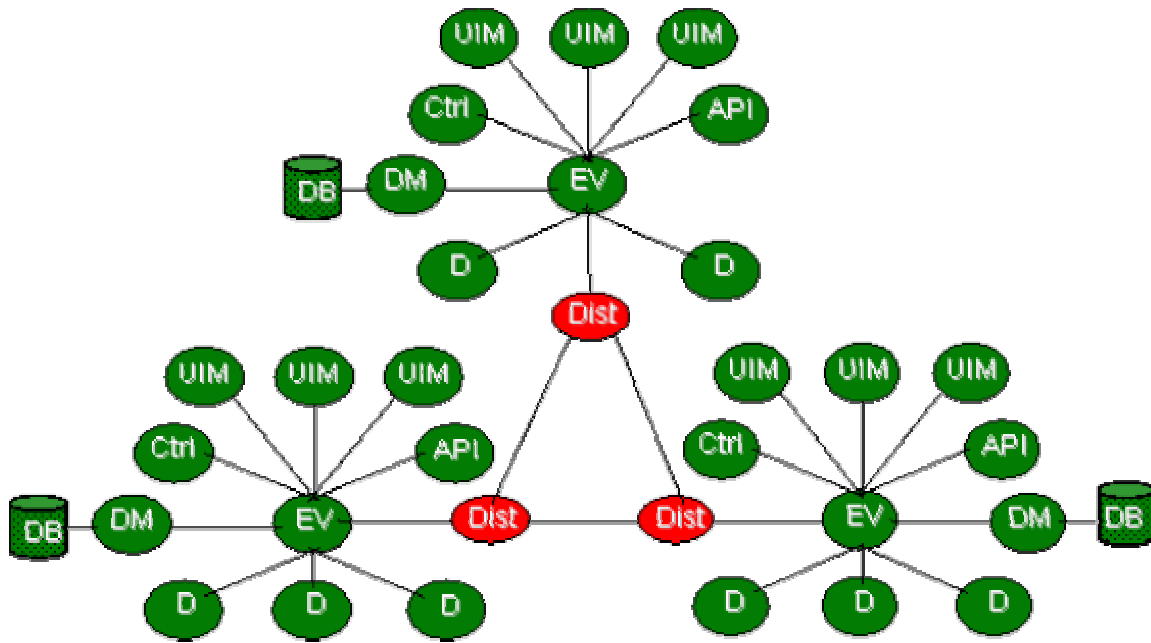


Figure 2: A Distributed System

For more information on building a distributed system then please see:

<http://itcobe.web.cern.ch/itcobe/Services/Pvss/FAQ/Questions/howDoIMakeADistributedSystemInV3.html>

3 Essential PVSS Concepts

3.1 Database Structure – The Data Point Concept

The device data in the PVSS database is structured as, so called, Data Points (**DP**) of a pre-defined Data Point Type (**DPT**). PVSS allows devices to be modelled using these DPTs/DPs. As such it allows all data associated with a particular device to be grouped together rather than being held in separate variables (as is typical in many SCADA systems.)

A DPT describes the data structure of the device (DPTs are similar to Classes in Object Oriented terminology) and a DP contains the information related to a particular instance of such a device (DPs are similar to Objects instantiated from a Class in Object Oriented terminology). The DPT structure is user definable and can be as complex as one requires and may also be hierarchical as shown in the example. In the example shown in Figure 3 we see a DPT representing a simple high voltage channel. This has a set of read (readings) and write (settings) parameters as well as associated display information - in this case the name of the panel to be used associated with it. Each of the folders (settings, readings and display) as well as the individual parameters (e.g. v0, vMon, panelName) are called Data Point Elements (DPEs) and are user-definable, i.e. the structure can be fully defined by the user .

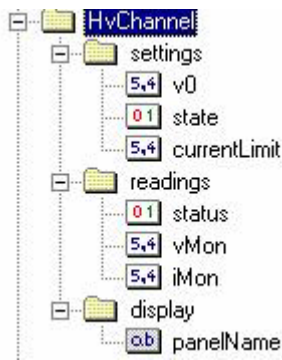


Figure 3: An Example of a Simple DPT

After defining the data point type, the user can then create data points of that type which will hold the data of each particular device.

In Figure 4 we see two instances of this DPT called Channel1 and Channel2 (DPs). For Channel1 we can see that for the vMon parameter a number of sub-elements are present (alert_hdl, archive, address, original, common and lock). These are known as ‘**configs**’ and enable specific behaviour to be configured for a DPE e.g. alarm handling or archiving as well as to hold attributes of that DPE e.g. its current value or its hardware address (**peripheral address config**).

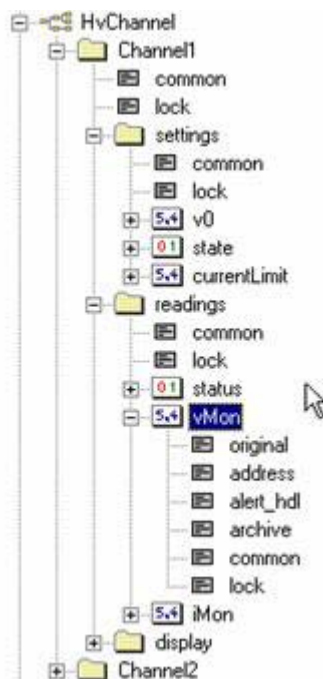


Figure 4: An Example of Two DPs Instantiated from the Example DPT Above

In this example the data point elements mapping to the device data are only of types float, boolean and string but several more data types are available. In particular **dynamic arrays** of the simple data types, like dyn_int, dyn_float, dyn_string, etc.

Dynamic arrays can hold a variable number of elements of the same type, i.e. the size of the arrays does not have to be defined by the user. Data point elements can also contain references to other data points or dynamic arrays of such references. The value of a DPE can also be derived as a function of other DPEs.

In order to access the data of a particular element of a data point, for example a channel's vMon, the user can address it as: **Channel1.readings.vMon** (corresponding to the dpName.firstLevelDpeName.secondLevelDpeName – this addressing scheme can also be followed for more complex DP structures with any number of levels in the DP structure).

The creation and modification of DPTs and DPs can be done either using the Graphical Parametrization tool (PARA), or programmatically using ctrl scripts.

3.2 Building User Interfaces

PVSS allows users to design their own user interfaces, in a “drag and drop” fashion. For that a special User Interface Manager – the Graphic Editor – can be started (NG for Windows and GEDI for Linux). An example is shown in Figure 5.

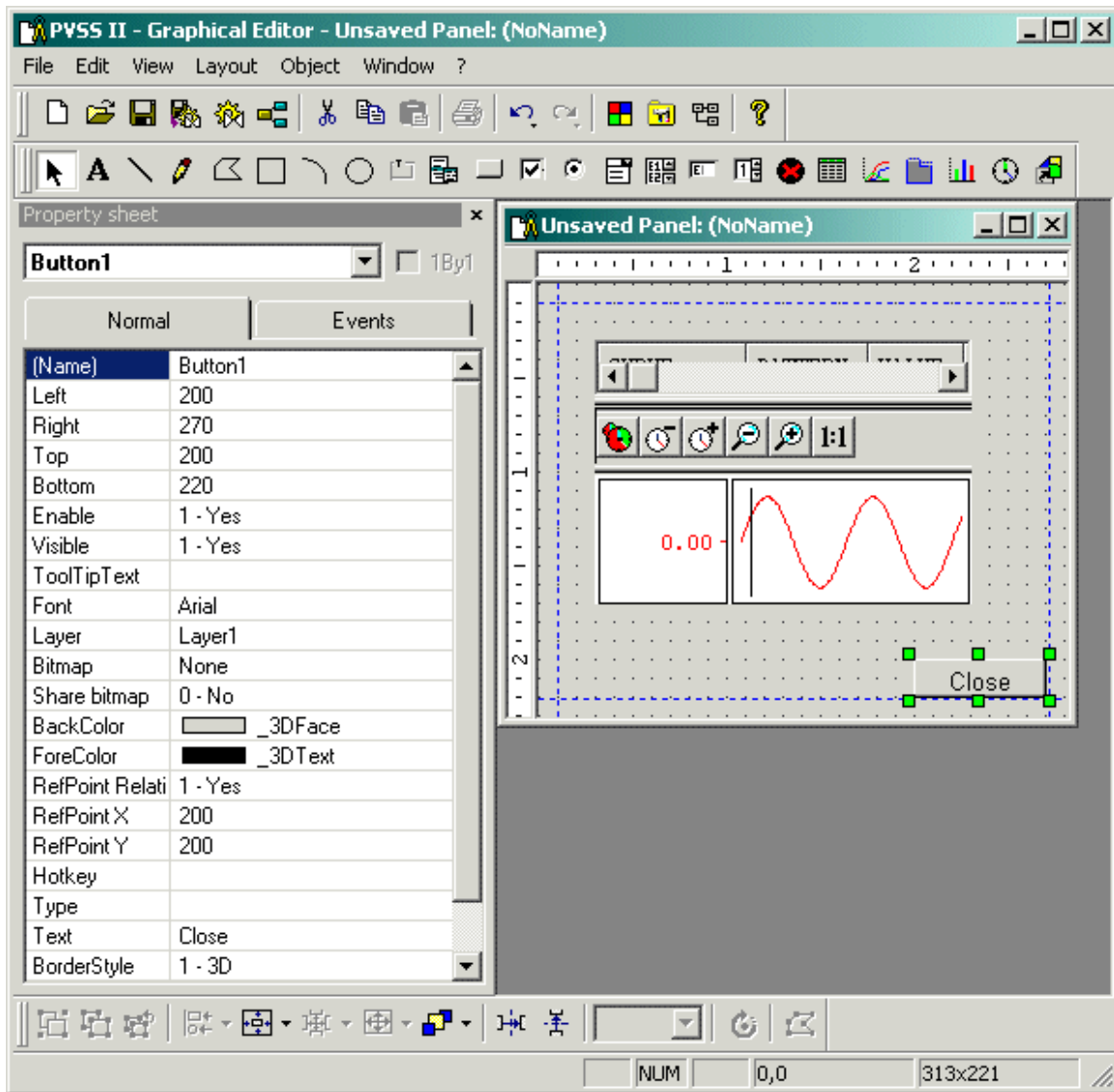


Figure 5: The Graphical Editor

By using the Graphic Editor (example above), the user can first design the static part of a panel, by placing widgets like buttons, tables, plots (called trends in PVSS), etc. Actions can then be attached to each widget. Depending on the widget type actions can be triggered on Initialization, user click or double click, text input, etc.

Actions are programmed using PVSS's scripting language, i.e. by means of CTRL scripts (see next section).

Figure 6 shows an example PVSS panel containing an example of a plot, a histogram and several other widgets like buttons and testfields.

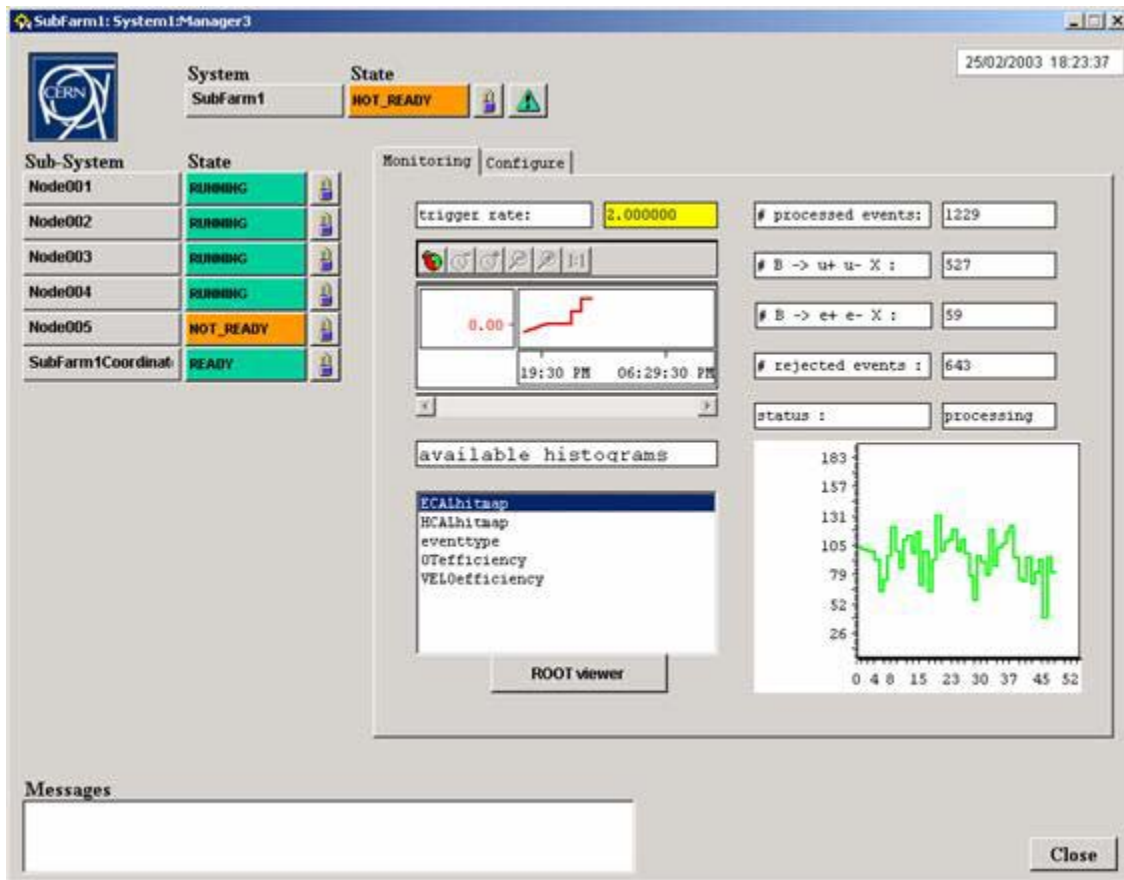


Figure 6: An Example Operations Panel

The same panel or graphic symbol may be required many times in a controls application, e.g. for controlling many instances of a particular device or collection of devices. To ease development PVSS provides the possibility to create a single symbol or panel and to use it many times. This is called a **Reference Panel**. Changes to this Reference *Panel* are inherited by all instances of the panel. Where actual variables, which vary between the various instances, are referred to in such a Reference Panel, these are replaced by variables called **\$parameters**. When the Reference Panel is instantiated these \$parameters are replaced by the appropriate actual variable names. This can be done either during development time when a Reference Panel is embedded into another panel or at run-time when the panel is opened. In this latter case the required variable names are passed as arguments when opening the panel.

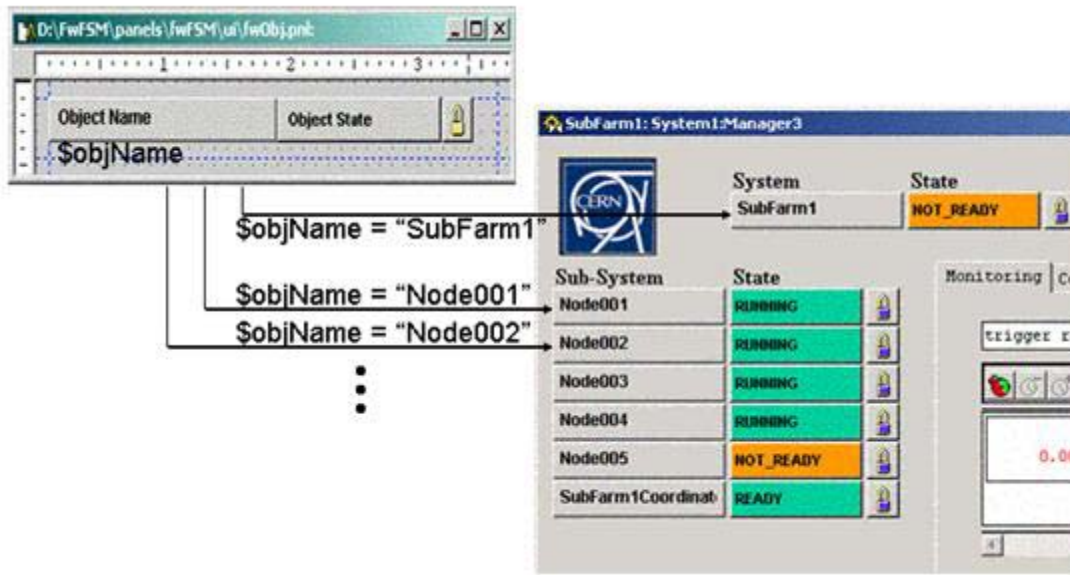


Figure 7: Use of a Reference Panel

In Figure 7 one \$parameter is used in order to pass the name of the object to be displayed. The name is then used inside the reference panel to get from the database the rest of the information related to the object, like its state, etc.

3.3 Using Control Scripts

PVSS Ctrl Scripts can be used in panels or as stand-alone processes. They provide the interface to the PVSS database. These scripts comply to the “C” syntax but they contain extensions, in particular new variable types, like “string”, and the provision for dynamic arrays.

PVSS provides a very large library of functions giving access to all PVSS functionality and more. There are functions for data point manipulation, for graphics, for file access, etc.

The three most important functions in terms of data point access are:

- **dpGet**(string dpName, <data_type> value)
Gets from the data base the current value of the data point element given as dpName. dpName is for example: "Channel1.readings.vMon"
value is a variable of the same data type as the data stored in the data point element, for example: int, float, dyn_string, etc.
- **dpSet**²(string dpName, <data_type> value)
Sets the current value of the data point element given as dpName.

² A function dpSetWait() also exists. dpSet() sends the message without waiting for an acknowledgement whereas dpSetWait() waits for an acknowledgement.

- **dpConnect**(string callbackFunctionName, string dpName)
Executes a callback function when the contents of dpName change.
the callback function gets called with two parameters: the name of the data point element (dpName) and the current value of the data.

The two most important functions in terms of graphical functionality are:

- **getValue**(string widgetName, string widgetProperty, <widget dependent data>)
getValue gets a property of a widget, like the font, the color, the size, the text is holds, etc. for example to get the background color of a button:
getValue("myButton","backCol",color); where color is declared as a string.
- **setValue**(string widgetName, string widgetProperty, <widget dependent data>)
setValue sets a property of a widget, like the font, the color, the size, the text is holds, etc. for example to set the text on a button:
setValue("myButton","text","Please Click Here").³

The complete set of functions is described in detail in the Online Help under the section "CONTROL"

As an example, in order to put in a panel a "Text Field" widget permanently showing the contents of a data point element (let's take our Channel1's vMon), one would attach to the widget's "EventInitialize" the following script:

```
main()
{
    // request that a callback be executed when contents of data point element vMon
    change
    dpConnect("updateCallback", "Channel1.readings.vMon");
}

updateCallback(string dpName, float newValue)
{
    // set this widget's text property to the new vMon value
    setValue("", "text", newValue);
}
```

Another example could be to read the value of a TextField widget and set it as Channel1's v0 when the user clicks on an apply button. For this one would attach to the "EventClick" of the Apply button widget the following script:

```
main()
{
    float newValue;
    // read the value of the v0 TextField widget
```

³ An alternative method for getValue and setValue is the so called dot notation
<Shape>.<Attribute>=value, e.g. mybutton.text="Please click here"

```

    getValue("v0TextField", "text", newValue);
    // set the value of data point element v0 to the new value
    dpSet("Channel1.settings.v0", newValue);
}

```

As mentioned before scripts don't have to be attached to panels, they can run in stand-alone mode. As an example we can permanently run a script that for example switches off the channel (by writing 0 to the settings.state element) if its vMon exceeds a certain value.

```

main()
{
    // request that a callback be executed when contents of data point element vMon
    change
    dpConnect("updateCallback", "Channel1.readings.vMon");
}

updateCallback(string dpName, float newValue)
{
    // Check if vMON is higher than 500 volt
    if(newValue > 500)
    {
        // Switch the channel off
        dpSet("Channel1.settings.state", 0);
    }
}

```

Any user functions that are repeatedly needed can be stored in libraries for use by panels and scripts. A script library is no more than a file containing the different functions and stored in a specific directory of a PVSS project.

It should be noted that PVSS provides wizards for adding standard behaviour to graphical objects, e.g. to set a DPE to a particular value, to change the colour of a graphical element based on the values of a DPE, etc.

3.4 Accessing Devices

In all the above examples we are assuming that our High Voltage Channel was connected to real hardware, i.e. we supposed that vMon was being updated and that when we wrote to v0 or state the data was sent to the channel. In order for this to be a valid assumption we now have to connect the data point elements in our database to real hardware (or software). This connection mechanism will depend on the type of the device that is being used.

PVSS provides several drivers to connect to various types of hardware or to different communication protocols. The type of protocol to use can be configured using the

Parametrization tool by selecting the **address config** of the relevant data point element and configuring it appropriately⁴.

3.4.1 The OPC Protocol

For our purposes the most important of the PVSS provided drivers which is of relevance, this is the OPC protocol driver, i.e. PVSS provides a generic OPC client. Other drivers of potential interest would be for CanBus and for ModBus over TCP/IP. As OPC is based on Microsoft's Com/DCOM technology the OPC client and server only run on Windows platforms.

Several industry HW manufacturers provide an OPC server. An OPC server is an abstraction layer between the real hardware and the user. For example a device, lets say a high voltage power supply could be read-out via CAN bus through a PCI card on a given PC. Instead of a user having to:

- read the power supply documentation to find out what registers and what bits in these registers he has to read or write in order to perform the operation he wants
- study the CANbus documentation and learn the CANbus protocol in order to be able to read or write the above registers via CAN
- Read the documentation of this specific PCI CANbus master card to see what libraries are available (assuming there are some) to perform CAN bus operations, in order to integrate them in its own program.

An OPC server implements all this and provides as a set of **OPC items** the data available from the hardware and the commands it can receive. In the example of a power supply it will provide parameters like: PowerSupply1.Channel1.vMon, etc.

OPC items are in a way very similar to our data point elements, so all we have to do is in the address config of the data point element, after choosing OPC as a protocol, define which OPC item it connects to.

Several vendors of HW commonly used in Physics experiments, are now providing OPC servers for their equipment. This is the case of CAEN, WIENER and ISEG which provide OPC servers for their various types of power supplies.

Also the Atlas ELMB (Embedded Local Monitoring Box) a radiation tolerant device for analog and digital IO provides an OPC server.

3.4.2 The DIM Protocol

OPC servers, although very easy to use, are quite complex to develop. So we do not encourage users to write their own OPC server for commercial equipment not providing one or for home made equipment.

⁴ It may be necessary to add the **address config** if this does not already exist

For this purpose the DIM protocol has been integrated into PVSS. DIM is also a client/server protocol and it provides a similar abstraction layer. Except that this time the equipment expert will have to write also the server part. DIM can be used both under Windows and Linux.

DIM servers provide DIM services which are similar to OPC items.

A PVSS DIM client provides the mechanism to connect to the server. As for OPC, the user will have to define the correspondence between data point elements and DIM services. Since DIM services can be structures containing different types of data, correspondence can be made between a DIM service and a full data point containing several elements. This correspondence can be made graphically or by using control scripts.

The full description of how to use the PVSS-DIM toolkit is available at:

<http://cern.ch/clara/fw/FwDim.html>

4 The JCOP Framework

The JCOP Framework is being developed in common for the four LHC experiments, with contributions from the experiments themselves and the IT/CO group at CERN.

This framework is based on PVSSII and provides guidelines and components, which can be devices or tools, for the different teams to build their control applications in a coherent manner.

The framework provides complete components for commonly used equipment. Complete meaning: any necessary OPC or DIM servers, the PVSS definitions in terms of data points types and data points, any control scripts and script libraries and all panels necessary to configure and operate the device.

At the moment the complete devices available in the framework are: CAEN high voltage power supplies, WIENER crates (power supplies and fan trays) and low voltage power supplies, ISEG high voltage power supplies, the ATLAS ELMB as well as generic devices to connect to Analog or Digital I/Os.

For these components the user will not have to know anything about PVSS or OPC, he just has to configure the system, i.e. specify what devices he has and their characteristics (number of channels, etc.), all this is done by pointing and clicking.

The Framework also offers tools for users to integrate their own devices, like the PVSS-DIM toolkit, and some reference panels and script libraries that can be used in order to build a new device. In addition, many FW standard panels and scripts provide functionality that can be used for other purposes in an application or as a model for building custom panels/scripts.

Other tools available include visualization tools, allowing users to define pages of plots, etc.

The complete documentation about the JCOP Framework and its components is available at: <http://cern.ch/itcobe/Projects/Framework/Documentation>

Another large component of the framework, although not completely integrated yet, is a tool to build hierarchies of components based on a Finite State Machine – the Controls Hierarchy. This tool allows users to define the desired behaviour of groups of components and to take actions and recover errors based on their states or on the states of external elements.

It also allows the various components to be partitioned in and out of the system in order to work in stand-alone mode. In the final system all devices in the experiment will be under the supervision of this tool.

Documentation concerning the Controls Hierarchy is available at:
<http://cern.ch/clara/fw/FwFsm.html>

5 PVSS Usage

5.1 Managing Projects

Two important aspects of PVSS which are important to understand when managing projects are those of the Installation and Project Directories. The Installations directory, as its name indicates, is where the PVSS files are stored. The directory has a defined structure of sub-folders. PVSS looks into specific folders to find the files it needs. Each project has its own directory, see Figure 8, which has a similar structure of sub-folders to that of the PVSS Installation directory. When looking for a particular file a PVSS application will always look in the project directory first and then the Installation directory afterwards. Hence if the project has a panel of the same name as a PVSS-delivered panel, then the application will use the project version.

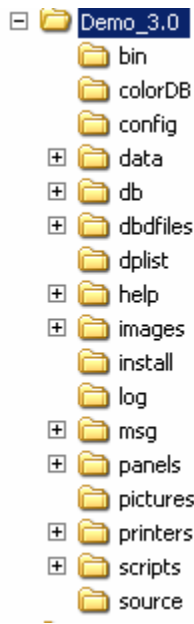


Figure 8: PVSS Project File Structure

5.2 *PMON*

This is a PVSS Manager which runs on both Windows and Linux in an identical fashion. It allows users to start, stop, configure and monitor PVSS Managers. The interaction with PMON can be achieved either via a Web Browser or via the PVSS Console, which is described below. Figure 9 shows PMON being accessed via MSIE. PMON can be started either via the Console, as a Service or from the command line.

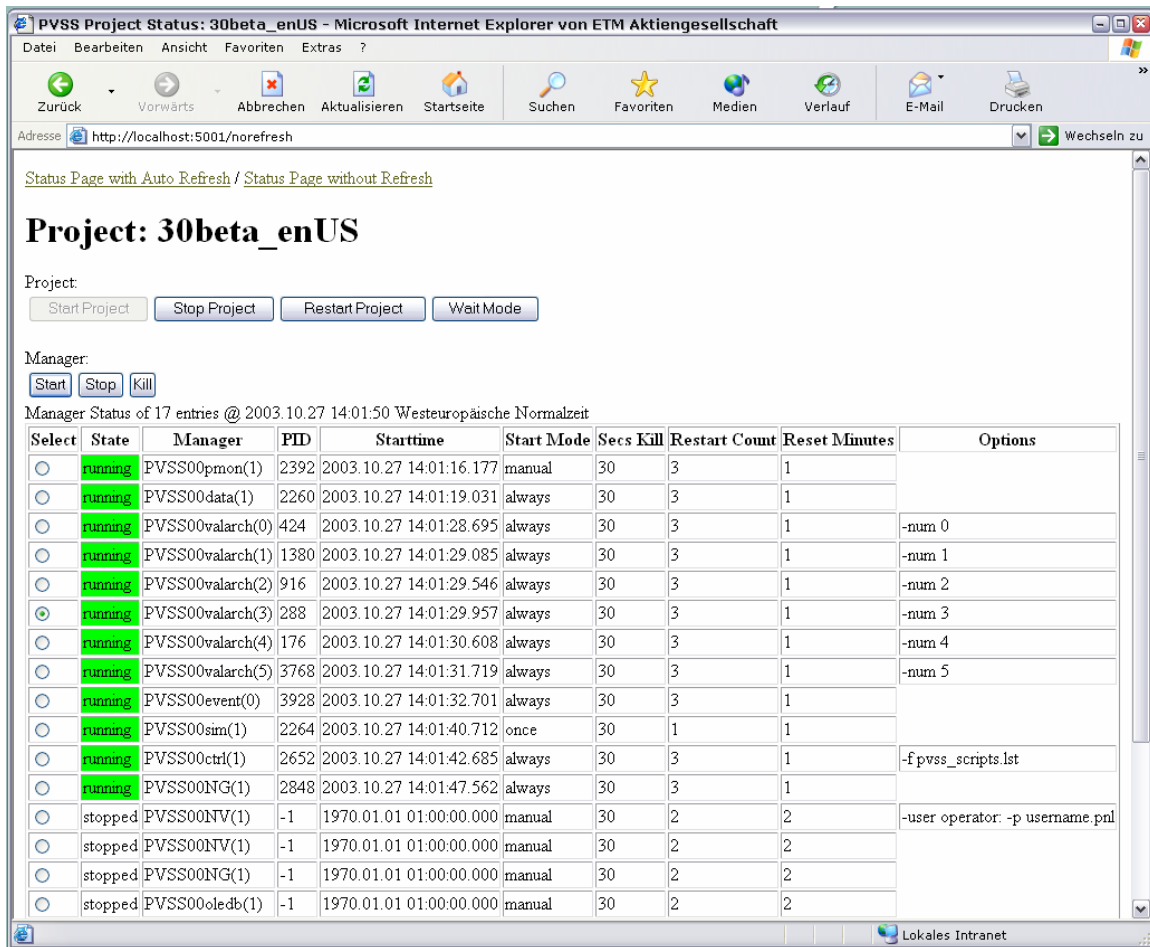


Figure 9: Accessing PMON via a Web Browse

5.3 PVSS Console

The PVSS Console, shown in Figure 10, allows users to interact with PMON in order to select a project to run and then to start, stop and configure the Managers to be run with that project. The figure below shows the PVSS Console, which is a standard PVSS panel which can run on both Windows and Linux in an identical fashion. From this panel a user can also upgrade a project to a more recent version of PVSS.

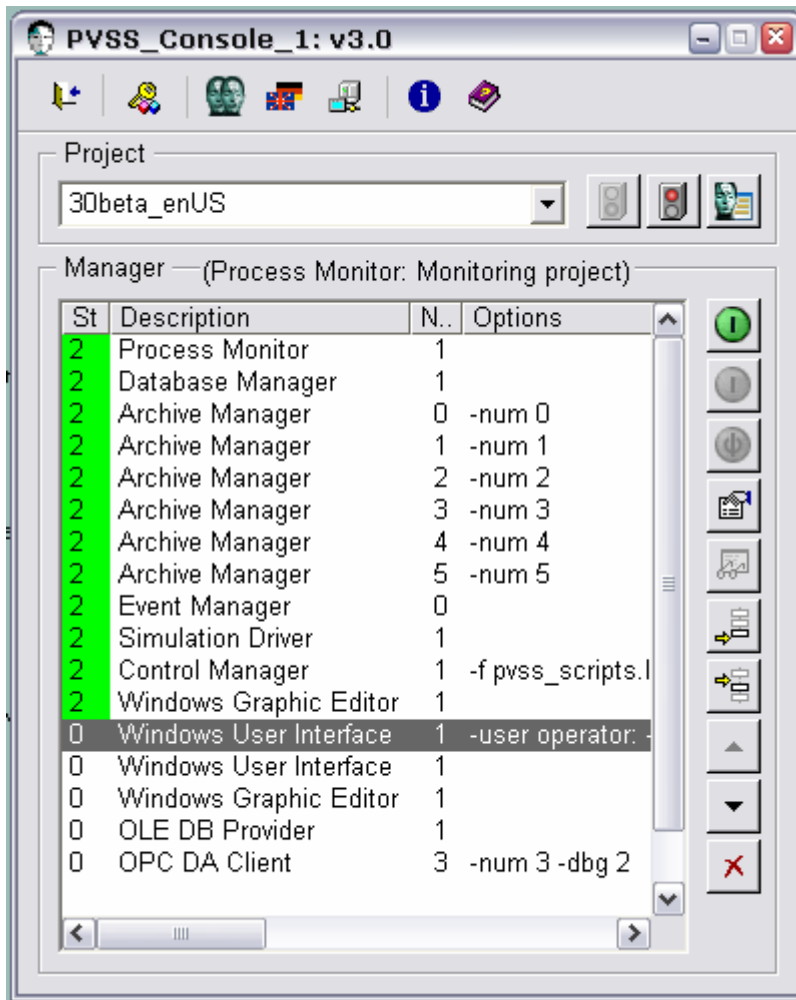
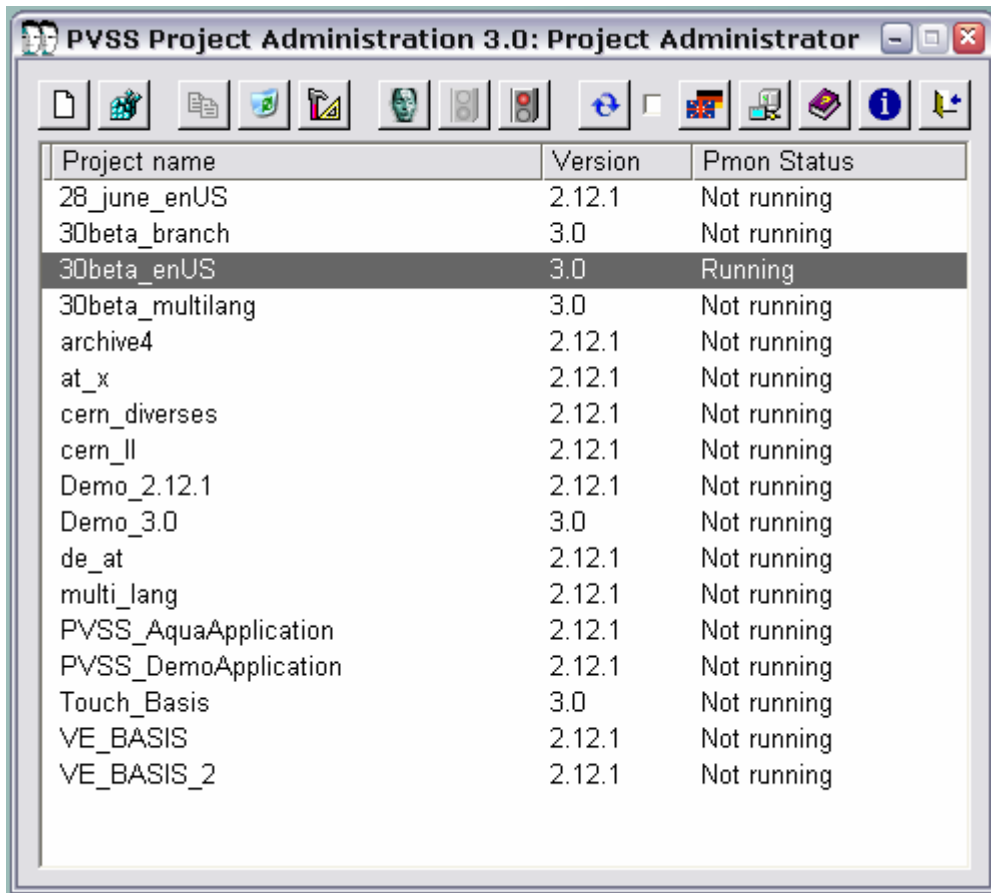


Figure 10: PVSS Console

5.4 Project Administration

The Project Administrator panel, as shown in Figure 11, allows the user to get an overview of all the PVSS projects which exist on that particular machine. It also gives the version of PVSS for which it is configured as well as the current state (running or not). For projects which are not currently running the user is able to perform actions such as copying the project, deleting the project, adding a new project as well as changing the configuration parameters for the project. From this panel a user can also upgrade a project to a more recent version of PVSS.



The image shows a screenshot of the 'PVSS Project Administration 3.0: Project Administrator' window. It features a toolbar with various icons for file operations, project management, and system status. Below the toolbar is a table listing several projects with their respective versions and monitoring statuses. The '30beta_enUS' project is highlighted as it is currently 'Running', while all other projects are 'Not running'.

Project name	Version	Pmon Status
28_june_enUS	2.12.1	Not running
30beta_branch	3.0	Not running
30beta_enUS	3.0	Running
30beta_multilang	3.0	Not running
archive4	2.12.1	Not running
at_x	2.12.1	Not running
cern_diverses	2.12.1	Not running
cern_II	2.12.1	Not running
Demo_2.12.1	2.12.1	Not running
Demo_3.0	3.0	Not running
de_at	2.12.1	Not running
multi_lang	2.12.1	Not running
PVSS_AquaApplication	2.12.1	Not running
PVSS_DemoApplication	2.12.1	Not running
Touch_Basis	3.0	Not running
VE_BASIS	2.12.1	Not running
VE_BASIS_2	2.12.1	Not running

Figure 11: PVSS Project Administration Panel

5.5 Log Window

The PVSS Log Window, as shown in Figure 12, displays all error messages originating from PVSS Managers. In case of problems when running PVSS this is the first to look for an explanation. The display indicates the source of the error message, i.e. which Manager, the time it was generated, the severity level and the error message number and text. A user has the possibility to filter the messages to be displayed.

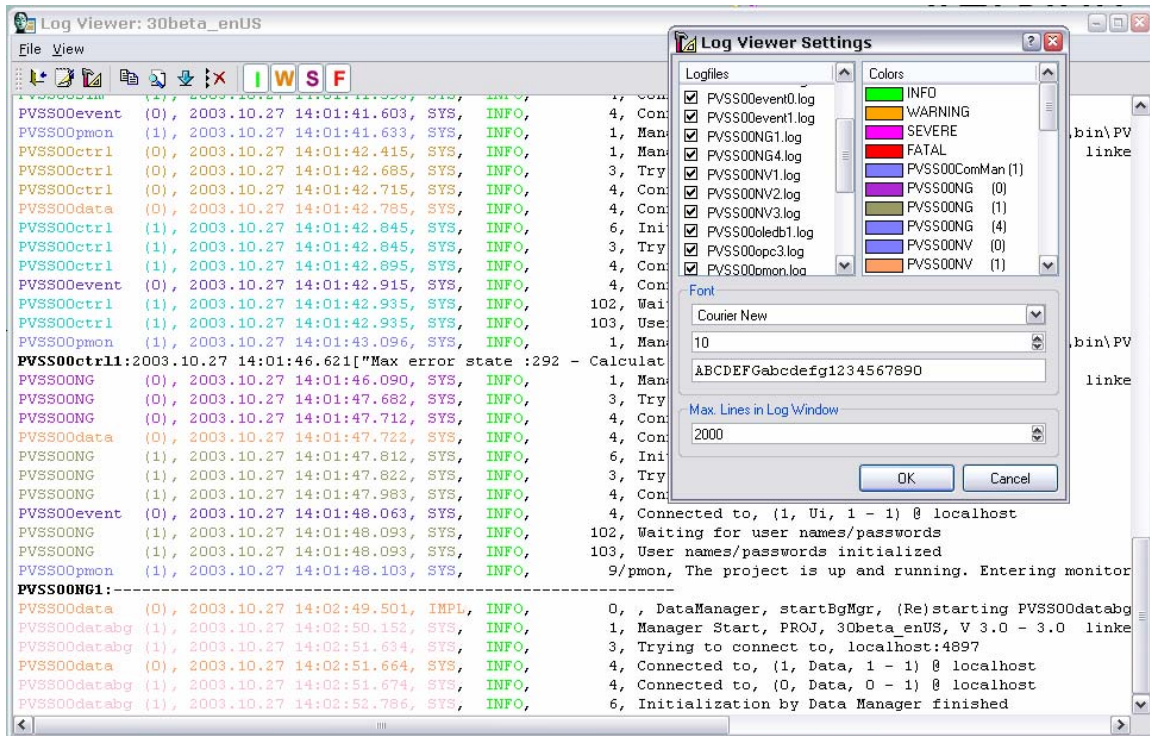


Figure 12: PVSS Log Viewer

6 Related Information

In order to install PVSS: <http://cern.ch/itcobe/Services/Pvss>

Tutorial on Starting Up PVSS:

<http://cern.ch/itcobe/Services/Pvss/ScadaLab/CSC/2001/lecture3.pdf>

General information about PVSS and Experiment's Control Systems:

<http://cern.ch/itcobe/Services/Pvss/ScadaLab/CSC>

JCOP Framework page: <http://cern.ch/itcobe/Projects/Framework>