Proview 4.2 Getting Started Guide

A step-by-step guide to set up a minimal Proview system

Jonas Haulin, Proview/R `<info@proview.se>`

Table of Contents

1. About this guide

This guide will take you through the steps of configuring, developing, simulating and running a small Proview project, on a single-computer system. The guide does not intend to be comprehensive. For detailed documentation, please consult the *Designer's Guide* or the *GE (graphical editor) Manual*. These documents are available at the Proview [http://www.proview.se] site.

1.1. Conventions used in this document

This document follows general conventions for content formatting, e.g.

- System items: `filename`, `user`, `ENVIRON_VARIABLE`
- Commands: To use the **pwrs** command, type **`pwrs`** at the terminal.
- Screen listing

```
bash$ cd src
bash$ ls -al
```

- Menu choices: Edit → Change value (**Ctrl-Q**).

In addition, the following Proview add-ons are used:

- Proview Environments are currently displayed in small caps or sans serif (PDF version), e.g. ProjectList.
- Proview Classes are currently displayed in boldface, e.g. **ProjectReg**.
- Attributes of Proview Classes are currently displayed in italics, e.g. *ObjectName*.
- Values of attributes are currently displayed in monospace, e.g. `56.4` in the case of numbers, or as “Demoprojects” in the case of strings.

2. Open the administrator environment

During installation, the user `pwrp` with password “pwrp” is added to the system. Log in as `pwrp` and use **pwra** at the prompt to start the Project Administrator. To start Proview as another user, just add the contents of the `.bashrc` file in the `pwrp` `home/` directory to that of your user.

The ProjectList administrator environment will open up in a new window. Switch to and from Edit mode with Edit → Edit mode (**Ctrl-E**). Make sure **NumLock** is not set on the keyboard, as Ctrl commands with NumLock enabled are not yet supported in Proview.

Tip

In some cases, you have to gain the Administrator privilege to enter edit mode. Login as administrator with the command **login /administrator**. (use **Ctrl-B**, to get a command line).

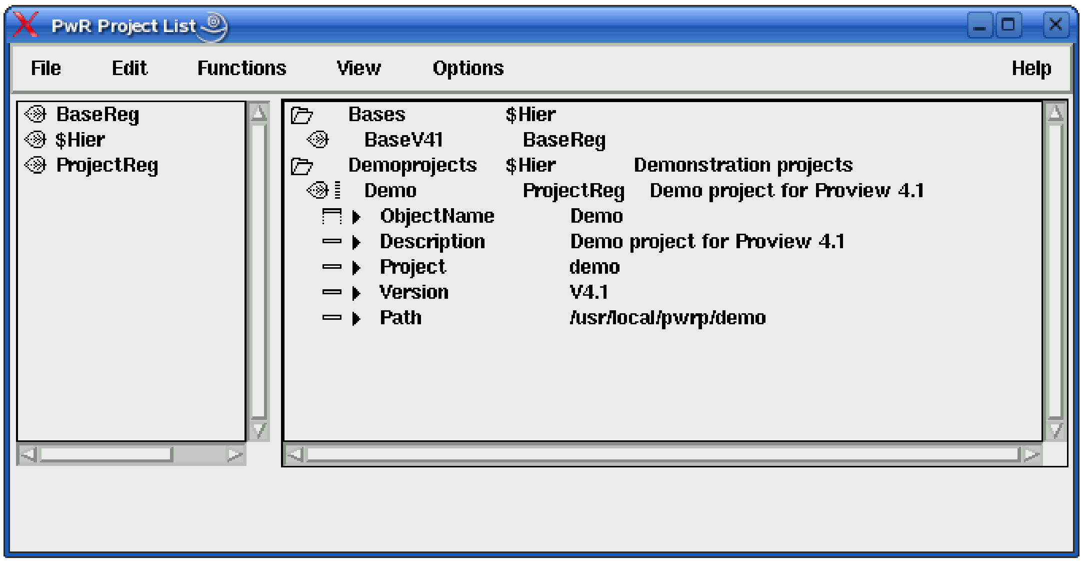
3. Create a project in the ProjectList

Once in Edit mode, an object palette will appear to the left in the window. The ProjectList at first only contains the “Bases” hierarchy. To create a hierarchy for the projects, select **Hier** in the palette, move the cursor to the “Bases” hierarchy and middle click. A new, empty **Hier** object is inserted below the existing one.

The hierarchy object is a container for arbitrary objects and it has two attributes: an *ObjectName* and a *Description*. To expand the object and expose the attributes, select the object and use the **right arrow** key. Now, to edit an attribute, select the attribute and use Edit → Change value (**Ctrl-Q**).

Use set adv

A more convenient way to edit an attribute is to give the command **set adv** in the Navigator (use **Ctrl-B**, to get a command line). Then object attributes may be open for editing with the **right arrow** key. In the case of an object with children, **right arrow** opens the next level of the hierarchy, and **Shift-right arrow** opens the object itself for editing.



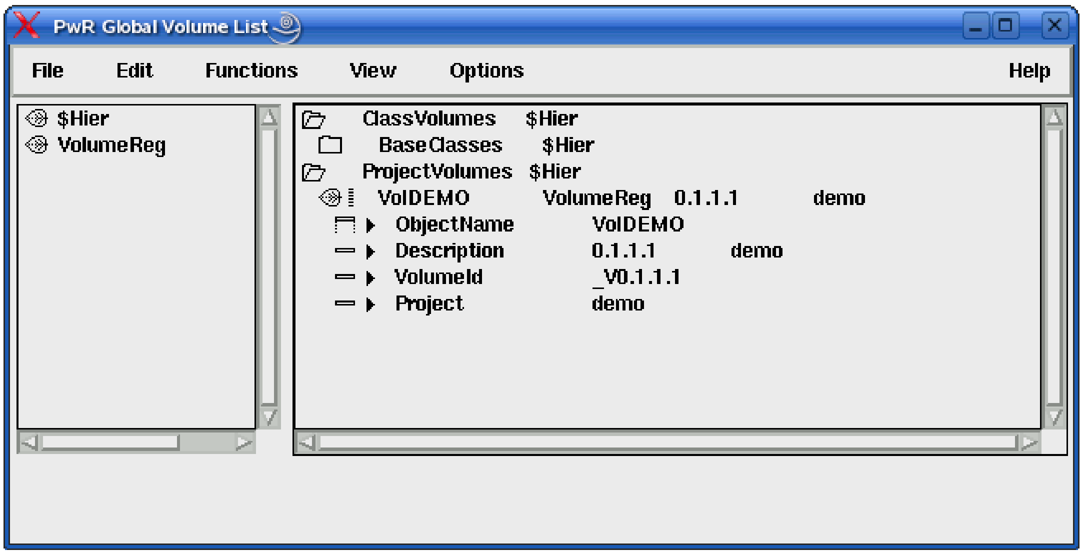
```
Bases          $Hier
  BaseV41        BaseReg
Demoprojects   $Hier          Demonstration projects
  Demo           ProjectReg   Demo project for Proview 4.1
    ObjectName     Demo
    Description    Demo project for Proview 4.1
    Project        demo
    Version        V4.1
    Path           /usr/local/pwrp/demo
```

The project list navigator.

We choose the name “Demoprojects” for our hierarchy. We then add a **ProjectReg** object to the “Demoprojects” hierarchy and name it “Demo”. We want to add the **ProjectReg** as a child and not as a sibling. To do this, middle click directly on the desired parent leaf, which is our “Demoprojects” hierarchy in this case. An object can be moved or copied in the hierarchy by selecting it and Right-clicking at the desired location. We need to define the *project* and *path* attributes. The project name is written in lower case. We use the name “demo”. The *path* attribute should just reflect the chosen project name. Save the changes to the ProjectList and exit Edit mode.

4. Create a volume in the VolumeList

In the VolumeList, select File → Open.. → GlobalVolumeList to open the VolumeList. Request Edit mode. In the "ProjectVolumes" hierarchy, add a **VolumeReg** object. We name our **VolumeReg** object "VolDEMO" and assign it to the right project by writing demo in its project attribute. A project may have several volumes assigned to it, but not the other way around. A unique *VolumeId* must be given. We choose `0.1.1.1`, see the *Designer's Guide* for a specification of the numbers. Save the changes to the volume list and go out of edit mode.



```
ClassVolumes   $Hier
  BaseClasses    $Hier
ProjectVolumes $Hier
  VolDEMO        VolumeReg  0.1.1.1      demo
    ObjectName            VolDEMO
    Description           0.1.1.1      demo
    VolumeId              _V0.1.1.1
    Project               demo
```

The volume list navigator.

4.1. Configuring the volume

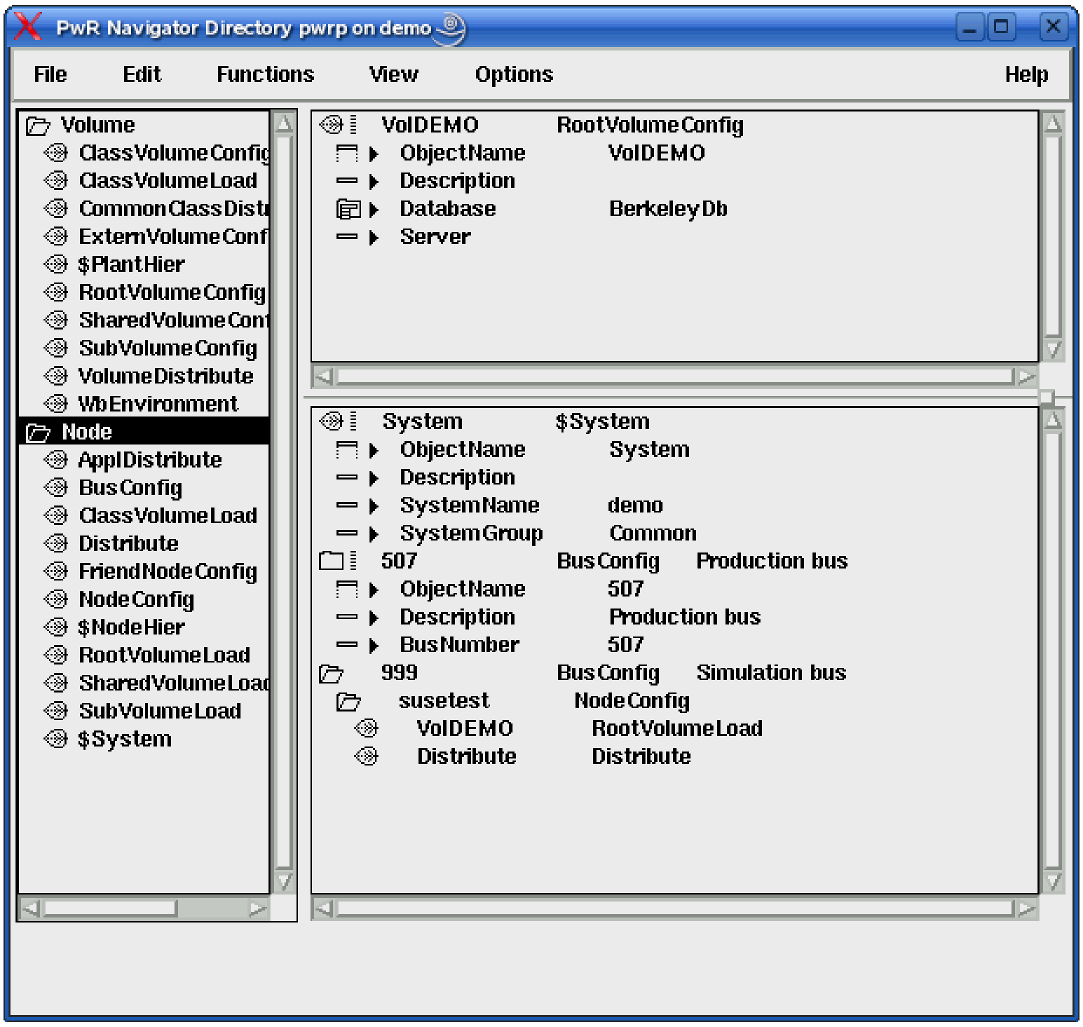
When a volume is created, open the Directory Navigator by right clicking the **ProjectReg** object in the ProjectList, and choosing Popup menu → Open Project... An alternative way is to type

```
bash$ sdf demo
```

The **sdf** command sets up environment variables for the project with default values. The Directory Navigator is then started with **pwrs**. In the Navigator, request Edit mode as before (**Ctrl-E**), and a two-window view (**Ctrl-W**). The upper window corresponds to Volume configuration, and the lower window to Node configuration. In the Volume window, we add a **RootVolumeConfig** object and name it "VolDEMO". In the Node window we add a **System** and two **BusConfig** objects.

In the **System** object, the *SystemName* attribute is the same as the project name, i e "demo". The *SystemGroup* attribute should be set to "Common", which gives us authorization to edit the project later.

The two **BusConfig** objects are for production and simulation, respectively. We name the objects after their bus numbers, choosing `507` for production, and `999` for simulation. These values must be set in the *BusNumber* attributes of the objects. To each of the **BusConfig** objects, we add a **NodeConfig** object. The **NodeConfig** attributes to be set are *ObjectName*, *NodeName* and *BootNode*, which are set to the name of the runtime machine in question. We only use the development machine `susetest`, and it is not possible to have separate buses for simulation and production. Therefore, we set up a dummy node on the production bus and put `susetest` on the simulation bus. Finally, the *OperatingSystem* (normally "Linux on x86") and *Address* (IP-address) of the node need to be specified.

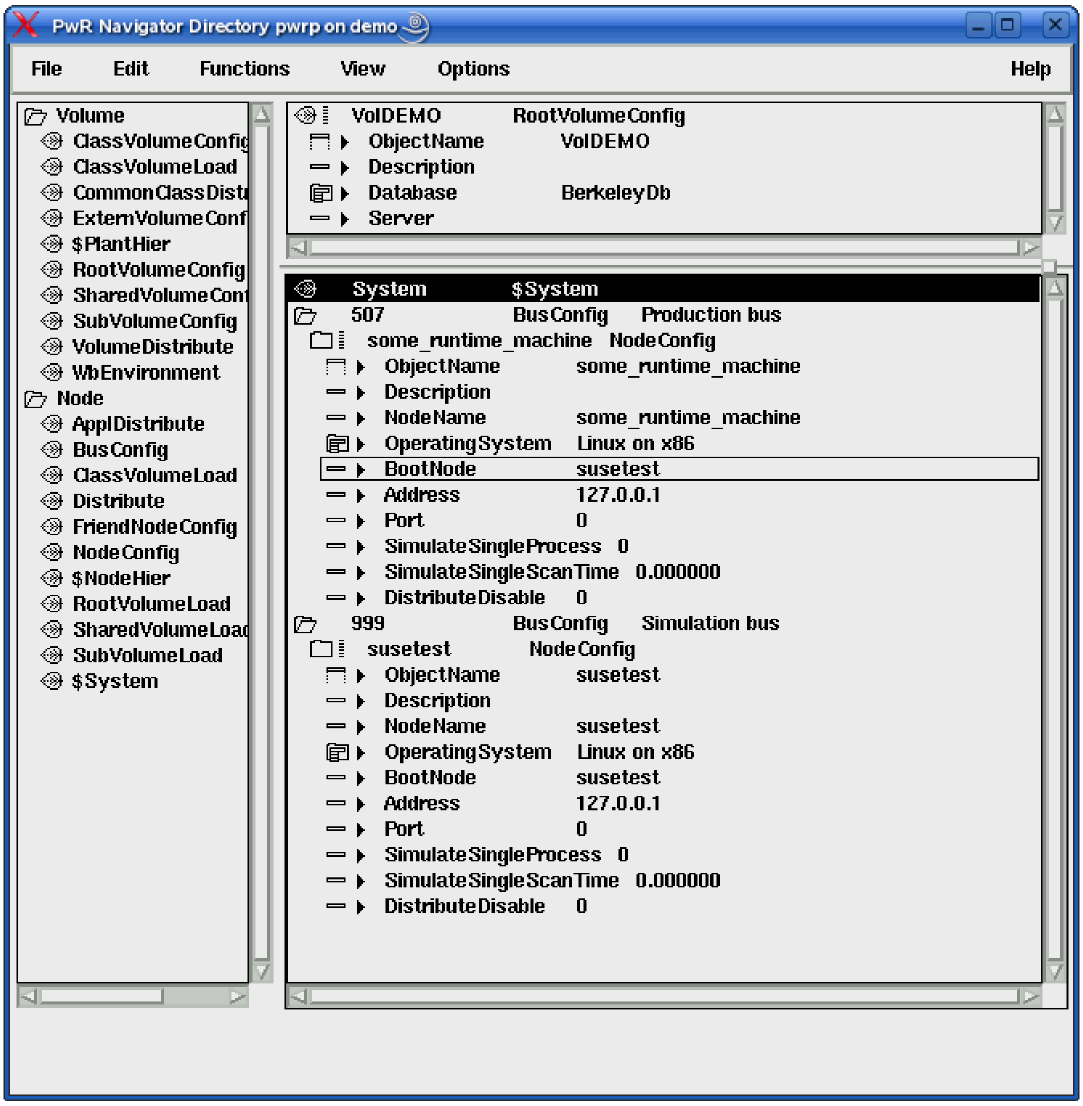


```
Volume
  ClassVolumeConfig
  ClassVolumeLoad
  CommonClassDistr
  ExternVolumeConf
  $PlantHier
  RootVolumeConfig
  SharedVolumeConf
  SubVolumeConfig
  VolumeDistribute
  WbEnvironment
Node
  ApplDistribute
  BusConfig
  ClassVolumeLoad
  Distribute
  FriendNodeConfig
  NodeConfig
  $NodeHier
  RootVolumeLoad
  SharedVolumeLoad
  SubVolumeLoad
  $System
```

```
VolDEMO          RootVolumeConfig
  ObjectName          VolDEMO
  Description
  Database            BerkeleyDb
  Server
```

```
System           $System
  ObjectName          System
  Description
  SystemName          demo
  SystemGroup         Common
507              BusConfig     Production bus
  ObjectName          507
  Description         Production bus
  BusNumber           507
999              BusConfig     Simulation bus
  susetest         NodeConfig
    VolDEMO          RootVolumeLoad
    Distribute       Distribute
```

The directory volume navigator.



```
Volume
  ClassVolumeConfig
  ClassVolumeLoad
  CommonClassDistı
  ExternVolumeConf
  $PlantHier
  RootVolumeConfig
  SharedVolumeConf
  SubVolumeConfig
  VolumeDistribute
  WbEnvironment
Node
  ApplDistribute
  BusConfig
  ClassVolumeLoad
  Distribute
  FriendNodeConfig
  NodeConfig
  $NodeHier
  RootVolumeLoad
  SharedVolumeLoad
  SubVolumeLoad
  $System
```

```
VolDEMO          RootVolumeConfig
  ObjectName          VolDEMO
  Description
  Database            BerkeleyDb
  Server
```

```
System           $System
507              BusConfig      Production bus
  some_runtime_machine   NodeConfig
    ObjectName            some_runtime_machine
    Description
    NodeName              some_runtime_machine
    OperatingSystem       Linux on x86
    BootNode              susetest
    Address               127.0.0.1
    Port                  0
    SimulateSingleProcess   0
    SimulateSingleScanTime   0.000000
    DistributeDisable     0
999              BusConfig      Simulation bus
  susetest              NodeConfig
    ObjectName            susetest
    Description
    NodeName              susetest
    OperatingSystem       Linux on x86
    BootNode              susetest
    Address               127.0.0.1
    Port                  0
    SimulateSingleProcess   0
    SimulateSingleScanTime   0.000000
    DistributeDisable     0
```

The directory volume navigator (again).

With each of the **NodeConfig** objects follows two children, a **RootVolumeLoad** object and a **Distribute** object. The *ObjectName* of the **RootVolumeLoad** object needs to be set. It is the same as the corresponding volume, “VolDEMO” in this case.

Save the changes and exit edit mode.

Tip

The process of configuring project and volumes can be automatated in a script, called from the command line in the Navigator. See the *Designer's Guide*, at the Proview [http://www.proview.se/] site.

5. Setting up plant and node hierarchies in the Volume Navigator

Open the Directory Navigator with **pwrs** from a terminal window (you may have to type `sdf` ***`demo`*** at the terminal first). Right click on the **RootVolumeConfig** object and choose Popup menu → Open volume.. to open the Volume Navigator. It is also possible to open the volume directly from the terminal, by passing the volume name as an argument to **pwrs**, i.e.

```
bash$ pwrs VolDEMO
```

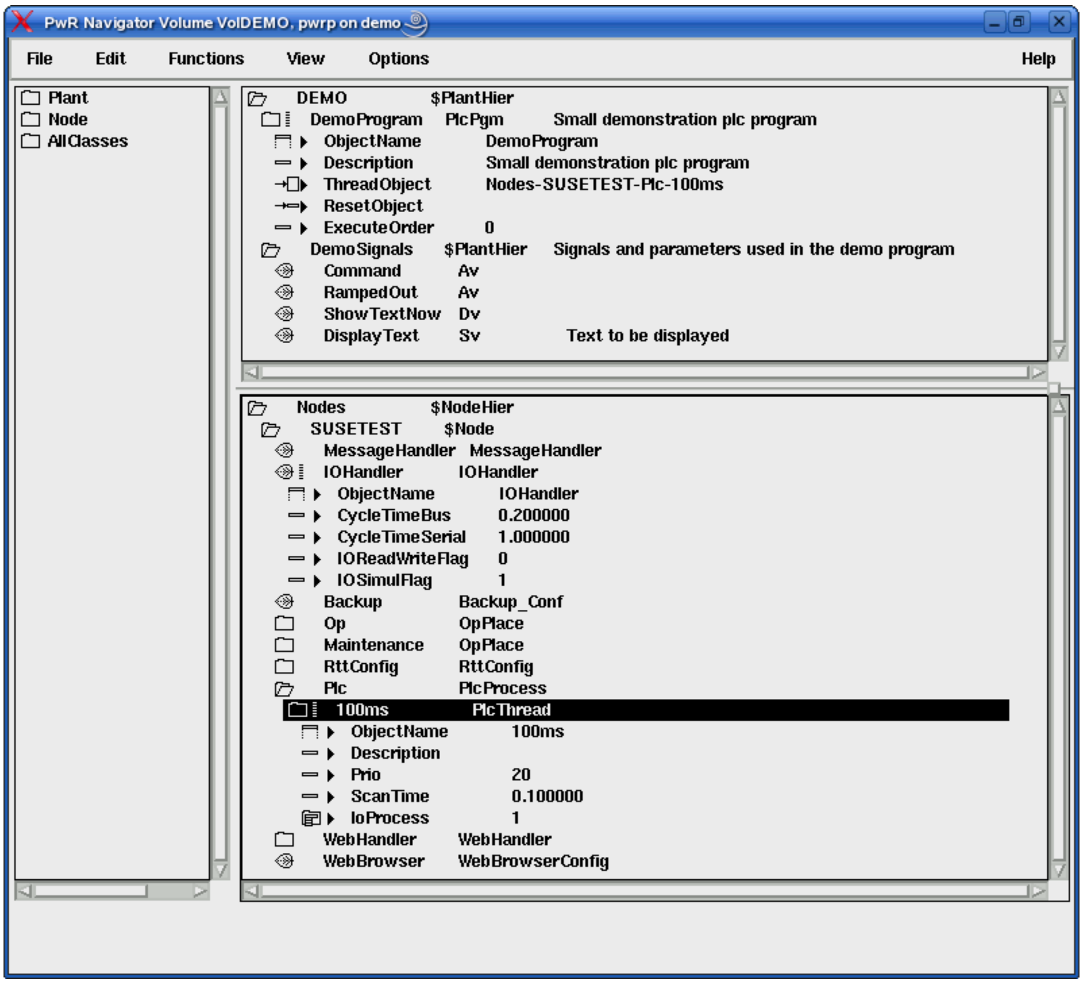
Enter Edit mode and choose a two-window view. Make sure that the *OperatingSystem* attribute is set correctly in File → Volume attributes...

Now, add a **PlantHier** object to the top window, and a **NodeHier** object to the bottom window. To the **NodeHier**, we add a **Node** object, and name it "SUSETEST". The plant hierarchy is a logical representation of the physical system to be controlled. It will contain the various signals and parameters needed, as well as the plc programs that perform the control operations. The node hierarchy is a representation of the hardware, with I/O-channels and system-level configurations. Proview currently supports the Profibus/DP fieldbus standard, as well as a number of less widespread QBUS I/O cards. To keep this demonstration example general, we will leave out I/O configuration, and use internal signals only.

5.1. Plant Hierarchy and Plc Programs

We will now set up the plant hierarchy, which is a logical model of our physical system, and create the plc programs that will perform the control operations. We start by adding some analog and digital signals (**Av** and **Dv**) to the plant hierarchy from the palette on the left-hand side. The palette is visible in Edit mode, only. Then we add a **PlcPgm** to the hierarchy (this is the plc program object). We need to connect it to the right **PlcThread** in the node hierarchy. Select the default "100ms" **PlcThread** under the **PlcProcess** object in the node hierarchy. Then connect it to the *ThreadObject* attribute of the newly created **PlcPgm** in the plant hierarchy. To do this, select the **PlcThread** and **Ctrl**-Double click on the *ThreadObject* attribute. Connections between other objects, e.g. I/O channels and signals, are created with **Ctrl**-Double click in a similar way.

We need to set the thread priority of the **PlcThread**, as the default priority of `0` will give poor real time performance. This is done in the *Prio* attribute of the **PlcThread**. A priority of `20` is suitable. Save the changes made to the volume, and exit Edit mode.

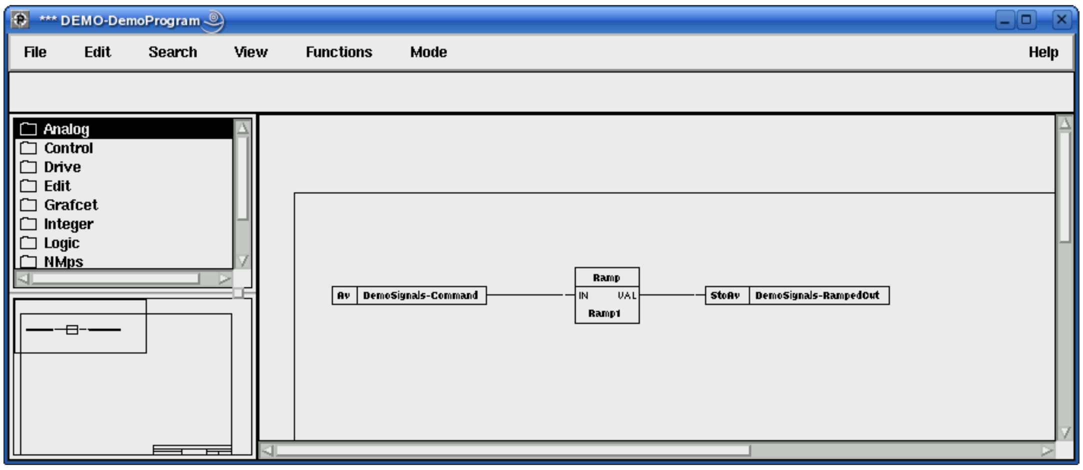


```
DEMO              $PlantHier
  DemoProgram     PlcPgm        Small demonstration plc program
    ObjectName                  DemoProgram
    Description                 Small demonstration plc program
    ThreadObject                Nodes-SUSETEST-Plc-100ms
    ResetObject
    ExecuteOrder                0
  DemoSignals     $PlantHier    Signals and parameters used in the demo program
    Command       Av
    RampedOut     Av
    ShowTextNow   Dv
    DisplayText   Sv                  Text to be displayed
```

```
Nodes             $NodeHier
  SUSETEST        $Node
    MessageHandler  MessageHandler
    IOHandler       IOHandler
      ObjectName      IOHandler
      CycleTimeBus    0.200000
      CycleTimeSerial 1.000000
      IOReadWriteFlag 0
      IOSimulFlag     1
    Backup          Backup_Conf
    Op              OpPlace
    Maintenance     OpPlace
    RttConfig       RttConfig
    Plc             PlcProcess
      100ms           PlcThread
        ObjectName      100ms
        Description
        Prio            20
        ScanTime        0.100000
        IoProcess       1
    WebHandler      WebHandler
    WebBrowser      WebBrowserConfig
```

The volume navigator.

To edit the plc program, make sure the Volume Navigator is not in edit mode. Then select the **PlcPgm** and choose Functions → Open program (**Ctrl-L**). The plc editor opens in a new window. Enter Edit mode. Now, add objects to the program by choosing from the left-hand side palette and middle clicking in the program area. We build a small example program consisting of a **Ramp** object, with analog input and output signals. Save the program and exit the editor.



The plc editor.

We now need to compile the program and create a load file and a boot file. This is done in the Volume Navigator. Choose Functions → Build Node in the menu. Check the outcome of the command in the terminal window where you started the Navigator.

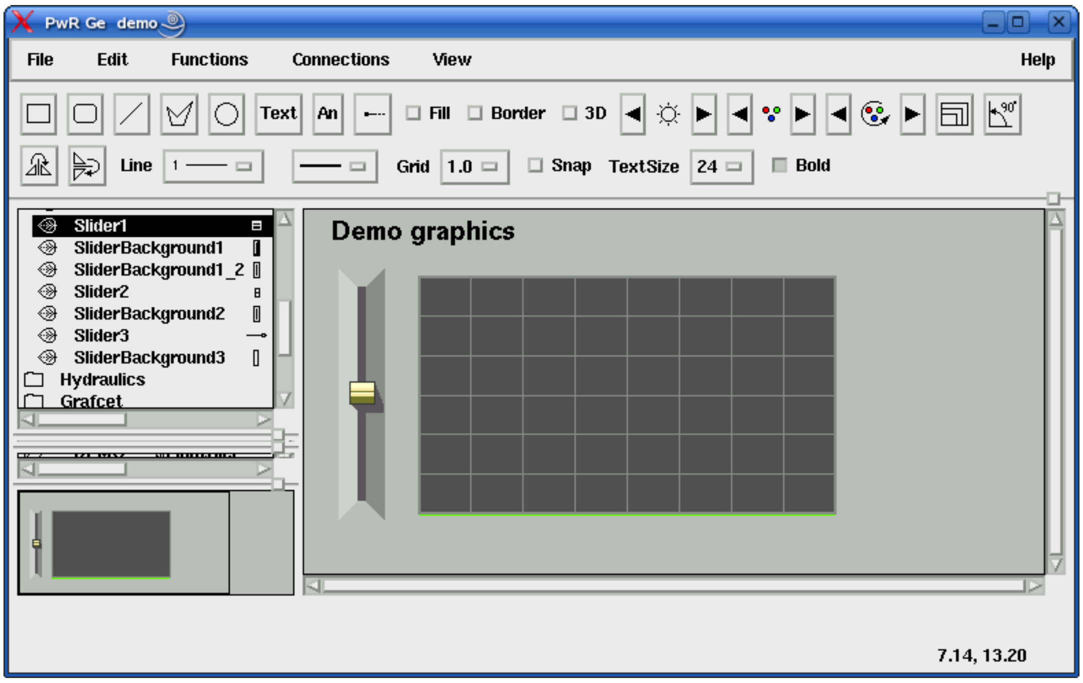
Note

A lock-file is created when opening Navigator to prevent other users from opening the same database. After an uncontrolled termination of the Navigator, the lock might remain, and has to be removed manually.

```
bash$ rm $pwrp_db/*.db.lock
```

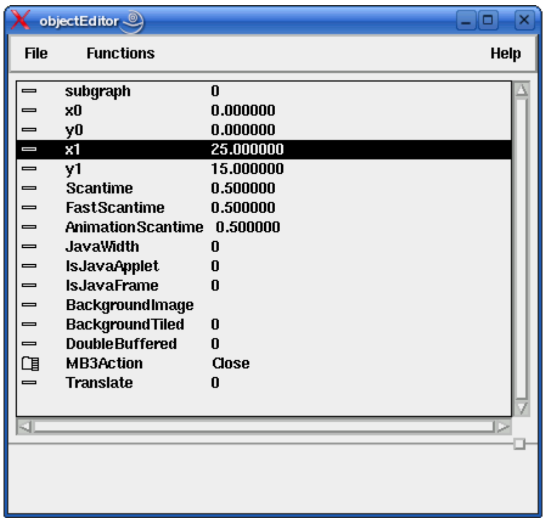
6. Process graphics and User Interfaces

To make a small GUI, open the Graphical Editor (GE) with Functions → Open graphical editor (**Ctrl-K**) To control the example system, add a slider with background and a trendcurve for the analog input and output. Open the object attribute windows, and connect the relevant attributes to the corresponding signals in the plant hierarchy. This is done by selecting the signal, and **Ctrl**-Double click the attribute.



The Graphical Editor (GE).

We need to set the size of the image in File → Graph attributes.... Place the cursor in the lower right corner of the image to read off suitable values for the x1 and y1 coordinates.



```
subgraph           0
x0                 0.000000
y0                 0.000000
x1                 25.000000
y1                 15.000000
Scantime           0.500000
FastScantime       0.500000
AnimationScantime  0.500000
JavaWidth          0
IsJavaApplet       0
IsJavaFrame        0
BackgroundImage
BackgroundTiled    0
DoubleBuffered     0
MB3Action          Close
Translate          0
```

The graph attributes window.

Save the process image file as `demo.pwg`. To use it in runtime we need to copy it from the default location in `$pwrp_pop/` to `$pwrp_exe/`. [1] When we have a lot of process graphics for a project the easiest way is to type

```
bash$ cp $pwrp_pop/*.pwg $pwrp_exe
```

[1]The initial **sdf** sets `$pwrp_pop/` and `$pwrp_exe/` to point to certain directories in the project tree. These are *`projectpath`*`/common/src/pop/` and *`projectpath`*`/x86_linux/exe/`, respectively.

7. Runtime

Now our program is compiled, and we have created a load file and a boot file for the volume. To start the runtime system on the development machine, type

```
bash$ rt_ini &
```

To stop the runtime processes, type **`rt_ini -s`**, or **`. pwr_stop.sh`** (while this holds for a development machine, the commands differ slightly on a runtime-only node, with a pwrrt package installed - see separate documentation).

Note

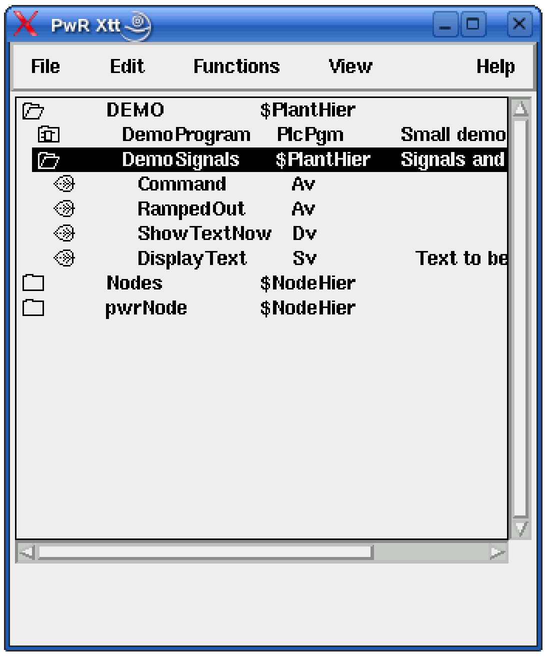
For the runtime system to work correctly, you may have to mount an `mqueue` on your system. This can be done by

```
bash$ su
bash$ mkdir /dev/mqueue
bash$ mount -t mqueue none /dev/mqueue
```

The `mqueue` thus mounted will disappear after reboot. For a permanent mount, edit the `/etc/fstab` file.

You may also want to control that the `$PWR_BUS_ID` environment variable is set up correctly (it should match the chosen bus number, which is `999` in this case).

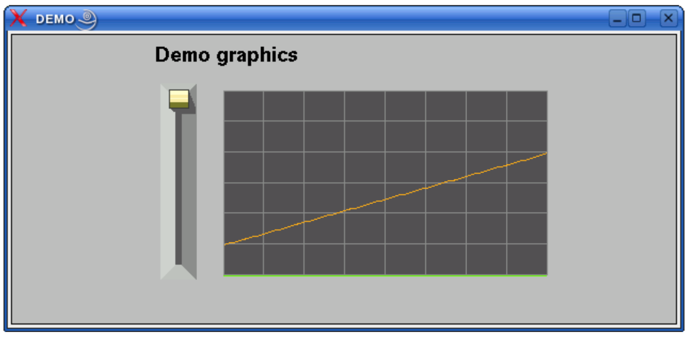
If everything goes well, we can now open the Xtt Runtime Navigator from the terminal with **rt_xtt**.



```
DEMO                $PlantHier
  DemoProgram         PlcPgm      Small demo
  DemoSignals         $PlantHier  Signals and
    Command             Av
    RampedOut           Av
    ShowTextNow         Dv
    DisplayText         Sv          Text to be
Nodes               $NodeHier
pwrNode             $NodeHier
```

The Xtt navigator.

To open the process image in Xtt, use **Ctrl-B** to get a command line in the navigator, and then type **`open graph demo`**.



The graph in the runtime environment.