Proview

Designer's Guide

5 May 2003

This guide gives information on the Proview Workbench; the environment used to configure, test and maintain Proview systems

SSAB Oxelösund

Overview of Proview

PROVIEW/R is a modern, powerful and general computer system. It contains all functions normally required for successful sequential control, adjustment, data acquisition, communication, supervision, etc.

The configuration of a PROVIEW/R system is done graphically, making the application adaptation simple, reliable, and flexible. PROVIEW/R is a distributed system, which means that the system consists of several computers, connected via a network . Via the network computers exchange data with each other. In this way, for instance, the measuring signals will be known on all the process - and operator stations of a PROVIEW/R system.

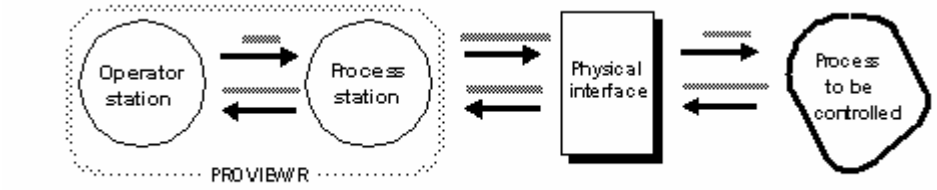
A PROVIEW/R system is defined with objects. Each object represents a physical or abstract entity in the system. The objects are structured in hierarchical tree structures, making it possible to describe the system in a structured way. The tree structure also imposes a hierarchical naming of all objects. By carefully choosing names when configuring a system, the full name of an object will identify its function, or role in the system.

The hierarchies are divided into two groups; one called the *Plant Configuration* , describing the logical view of a system; and one called the *Node Configuration* , describing the physical view of the system. The configuration of these two parts can be done independently. Eventually the two parts are connected to each other.

To configure a system you use the PROVIEW/R Workbench. The workbench comprises a permanent database and a number of tools to help you configuring the objects needed to define the system. From the workbench you create a runnable system as well as documentation about the system.

The purpose of PROVIEW/R is to help you creating automated systems. Suppose you have a process that you wish to control, PROVIEW/R helps you creating the control system for this process. When the system is created, you will find that you have also created the documentation for the system.

Automated system



Briefly described, PROVIEW/R is often connected to the process by a process station via a physical interface. The process station controls the process, depending on what input it receives. The operator at an operator station receives information, visual, acoustic, or other, on the process. The operator can give orders to the process station, and in that way control the process.

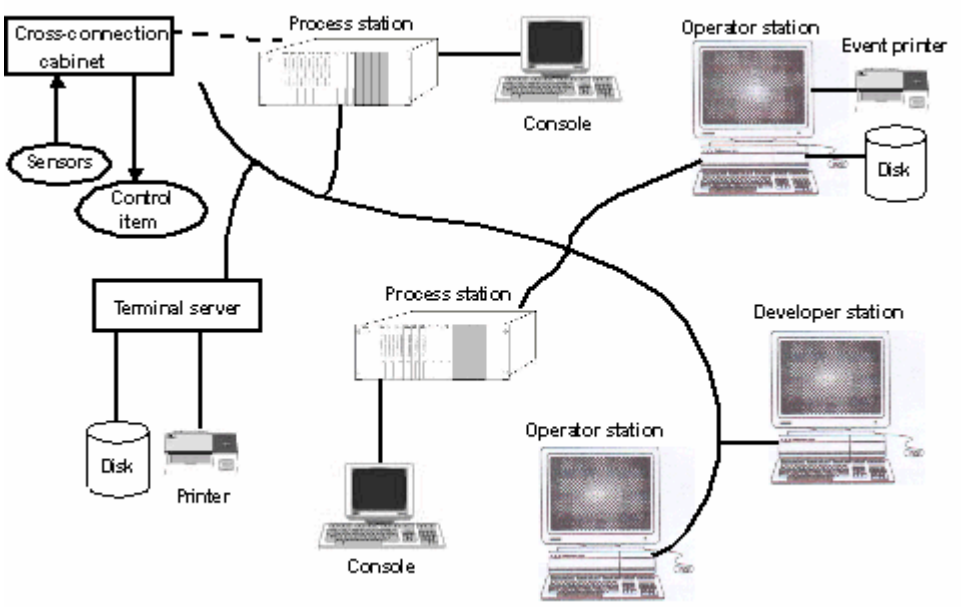
Hardware

A PROVIEW/R system comprises a number of computers, or stations , which can be divided into three groups according to their main function in the system:

- **Process station**
  Executes the PROVIEW/R Process Environment.
- **Operator station**
  Executes the PROVIEW/R Operator Environment
- **Developer station**
  Executes the PROVIEW/R Development Environment

The process station and the operator station can together be called the target environment .

Hardware of a PROVIEW/R System



Each type of environment and thus each type of station handles specific tasks in the PROVIEW/R system. Often a PROVIEW/R system comprises a number of process stations and operator stations, and at least one developer station. It is, however, possible to run more than one environment in one station. The minimal PROVIEW/R system is one station running all three environments.

We will first take a brief look at the hardware components in a PROVIEW/R system, some of which are shown in figure *Hardware of a PROVIEW/R System*, and the tasks they perform. We will then give a deeper description of each environment.

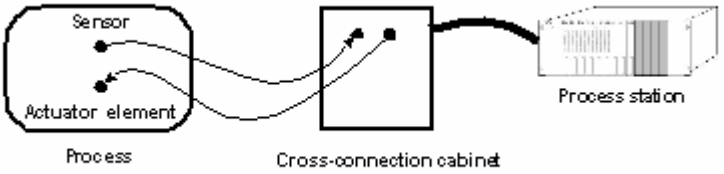
Process Station

A process station executes the PROVIEW/R Process Environment. It is often a rack-mounted PC but it could also be a workstation.

The operating system used on a process station is Linux or LynxOS.

I/O cards are connected via both buss- and serial interfaces.
The process station is connected to the sensors and control elements of the process via cross-connection cabinets.

A Process Connected to the Process Station



Operator Station

An operator station is a workstation executing the PROVIEW/R Operator Environment. The operating system used on operator stations is Linux.

An operator logs on to an operator station as a certain user or user category. The system designer specifies a unique environment for each individual user. The operator station can accommodate one or several operator environments at the same time.

Via the network the operator has access to information from the process stations as well as from other operator stations. From the operator station the operator can send orders to a process station and in that way control the process directly or indirectly.

Developer Station

A developer station is a workstation executing the PROVIEW/R Developer Environment. The operating system used on a developer station is Linux.

I/O

Input and output of data are usually called I/O. If the process cards are positioned in the rack of the process station, we have a local I/O. However, the process cards can also be positioned in distributed racks, which are connected with process stations via cables, and are then called distributed I/O.

Network

For network communication PROVIEW/R Ethernet.

Database Structure

The PROVIEW/R database consists of objects . These objects defines the different physical and logical entities needed to describe and operate a system.

Object

An object is an entity able to save a state (information) and which offers operations to either examine or affect this state. An object is an abstraction of a physical entity or of something in the domain of a problem or its implementation, reflecting the capabilities of a system to keep information about it, interact with it, or both.

Class

In a system there will be a number of objects. Some of these objects will have common characteristics and we can group these objects according to their characteristics. Such a group represents a class . In order to describe all objects having similar behaviour and information structure, we identify and describe a class to represent these objects.

A class represents a template for several objects and describes how these objects are structured internally. Objects of the same class have the same definition both for their operations and for their information structure.

A PROVIEW/R system includes about 400 classes, and can be extended with user defined classes. All classes are defined by objects, so-called meta objects, stored in the database and thus making it possible, in a running system, to obtain information about all classes.

Most classes are described in the PROVIEW/R Objects Reference Manual .

Attribute

To store information, objects use attributes . To each object we can associate several attributes. Each attribute has a type , which can be a primitive data type, such as an integer or a string, but it can also be a composite data type which is more complex and that is especially defined. An attribute has a name, a dimension, and a type.

Instance

An object that belongs to a certain class is called an instance of that class. The term object and the term instance are therefore often used as synonyms.
An instance is an object created from a class. The class describes the behaviour and information structure of the instance, while the current state of the instance is defined by the operations performed on the instance.

Objid - Object Identity

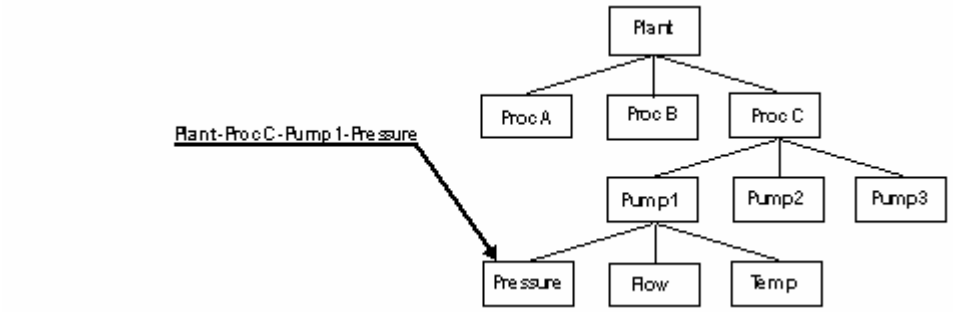
Each object defined in PROVIEW/R Workbench has a unique 64 bit identity, called Objid . This identity exists throughout the lifetime of the object.

Objects created in the runtime environment gets an Objid lasting to the next reboot.

Object Naming and Hierarchy

Each object must be given a name , which can be up to 31 characters long. Valid charcters are all alphabetical characters (including diacritical characters as "å","auml;" and "ouml;")and digits. Both uppercase and lowercase characters may be used to form readable and meaningful names. Naming in PROVIEW/R is however not case sensitive, i.e. the names "Pump1" and "pump1" will refer to the same object in PROVIEW/R.
All objects are members of a hierarchical naming structure. When an object is created it is placed somewhere in a hierarchy. It gets a parent and siblings (or sometimes called brothers). The full name of an object is the full name of the parent and its own name, separated by a dash. The full name can be up to 79 characters long.
E.g. the object pointed at in figure See Object Naming is Plant-ProcC-Pump1-Pressure.

Object Naming



To Summarise

Each object instance:

- is an area of information storage, structured in attributes
- is member of a class defining the characteristics of the object
- is member of a hierarchical naming structure
- has a unique 64 bit identity, Objid
- resides in a volume

The Workbench Database

When you configure a PROVIEW/R system you define objects which are stored in a database, called the Workbench database. The PROVIEW/R Workbench comprises a number of tools to help you defining these objects.

From the Workbench database you create load files which are read by the runtime environment at startup.

The Real-Time Database (Rtdb)

When running a PROVIEW/R system, a volatile database, called the real-time database is created. Initially the real-time database contains all the objects defined to be there by the Workbench, but objects can be created, deleted, renamed, moved, and changed dynamically.

Distribution of Objects

The real-time database is shared by all nodes in a PROVIEW/R system:

- Objects are defined to reside on a specific node.
- All objects are available on all mounted volumes.
- All changes to the database are distributed to all nodes.

Each object has a header and a body. The object header contains information on the qualities and status of the object, its residence, as well as its relations to other objects. The object body contains the object data, i.e. the attributes.

Object attributes are available to all nodes by means of:

- Direct manipulation (read and write).
- Subscription (periodical update, read only).

Note! From a PLC-program you can only write to objects that resides at the node in which the PLC-program runs. See below about object residency.

Global Data Handler (GDH)

Any application can access the real-time database through an application interface called the Global Data Handler (Gdh). It can for example:

- Create and delete objects.
- Examine relations between objects.
- Read and write attributes.
- Establish subscriptions.

The PROVIEW/R Programmer's Reference Manual provides guide and reference information about Gdh routines.

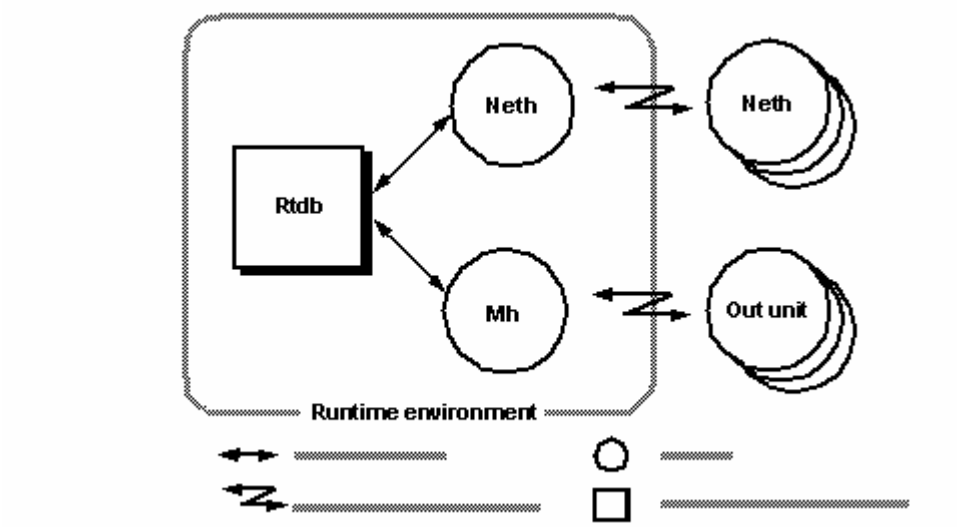
Object Residency

When you start the runtime environment in one node, the objects defined to reside at that node will be known immediately. Objects residing at other nodes will be known when contact is established with those nodes.

Runtime Environment

The runtime environment handles the distributed real-time database, and distributes messages to all out units known in the system.

The Runtime Environment



Each node participating in a PROVIEW/R system executes a runtime environment.

At startup the runtime environment reads load files specific to the node, and in that way populates the the real-time database with objects defined to reside at that node. The Net Handler contacts Net Handlers on other nodes and in that way all mounted volumes will be known.

The Message Handler is responsible of distributing messages on events detected on the own node, to all out units in the system. An out unit is e.g. the Operator dialog and the event printer.

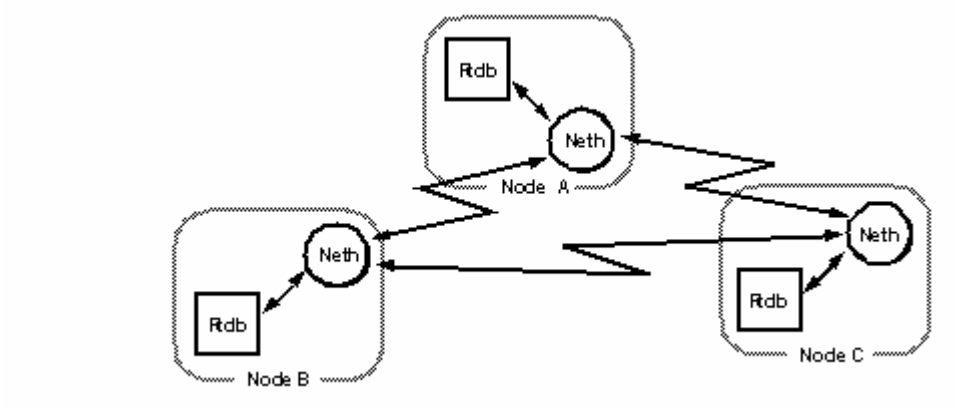
Net Handler

The Net Handler handles the distributed database and is responsible of:

- Keeping the distributed database together as one unit.
- Dynamic object creation and removal.
- Reading from and writing to objects in other nodes.
- Handling requests for subscription on objects from other nodes within a PROVIEW/R system, and to transfer such information.

Each node in a PROVIEW/R system has a Net Handler and a part of the distributed database.

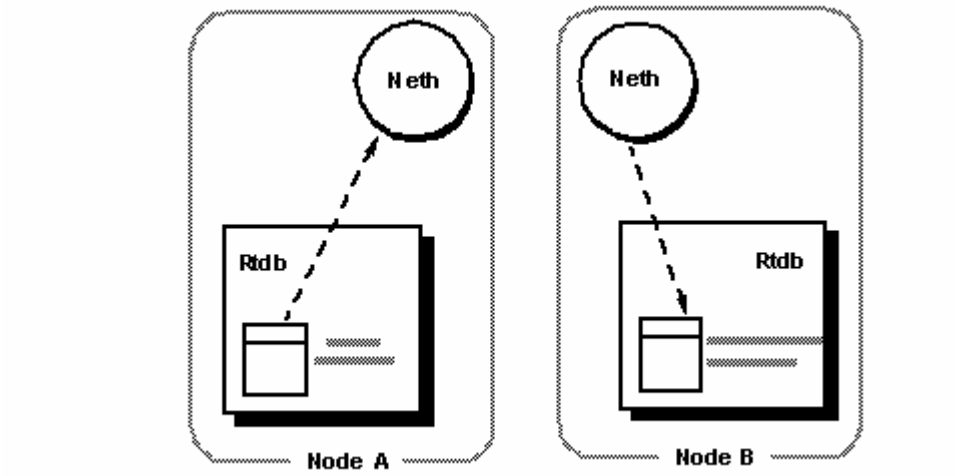
Net Handler and the Distributed Database



Subscriptions

For cyclic updating of an object, you use a subscription . For instance, to update the dynamics of a plant graphics at an operator station.
Figure See Subscription illustrates how a subscription is set up: An application on node B wants to subscribe to the object ProcA-Test residing on node A. A copy of the object will be created in the database of node B, and data will then be copied periodically. Should the contact to node A be broken, the updating will continue as soon as contact to node A is re-established.

Subscription



Message Handler (Mh)

The Message Handler (Mh) is used to handle alarms and messages.

Mh handles the supervisory objects (DSup, ASup and CSup objects) and checks whether an alarm status exists, whether return to normal conditions has taken place, or whether an acknowledgement has been given. One supervision object for each alarm limit is required.

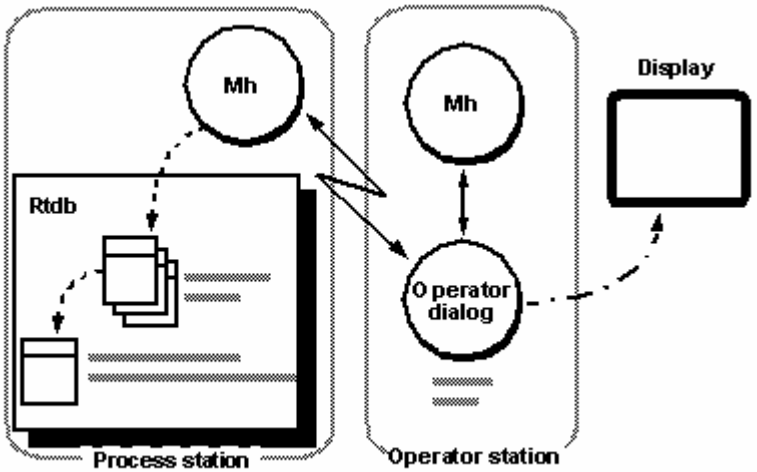
When an event has been detected, a message is sent to the out units, asking for that particular event. The individual out unit can specify, from which object hierarchies messages should be received.

Figure *Message Handler* illustrates what happens, when the supervised analog signal object ProcA-Tempin exceeds one of its alarm limits. The Message Handler in the process station detects that the Proc-Tempin has exceeded an alarm limit and sends a message to an out unit, in this case an operator dialog, which present the alarm on the operator's display terminal.

Also applications can be connected to Mh for sending alarm and information messages to out units.

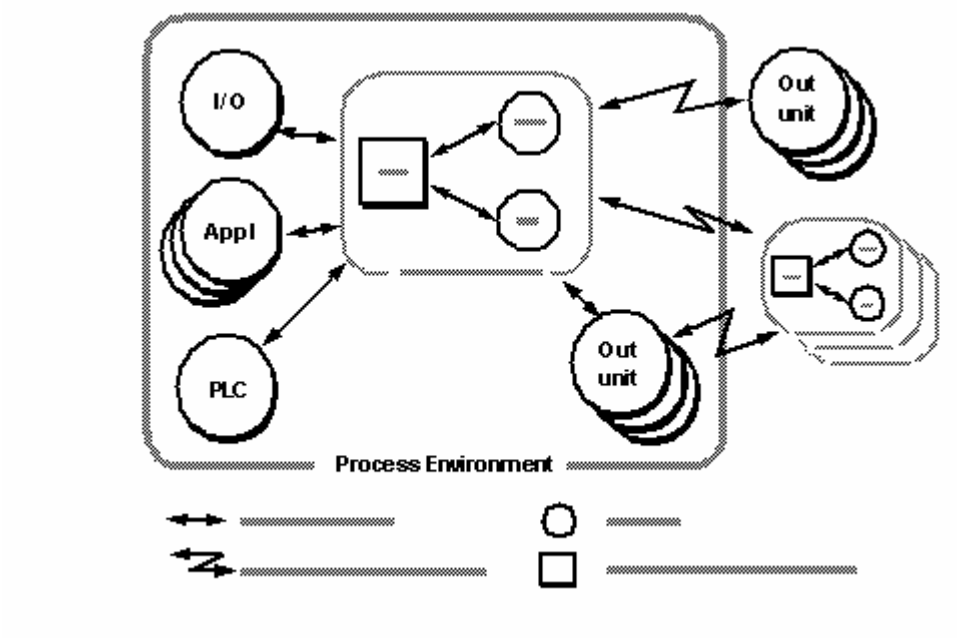
The PROVIEW/R Programmer's Reference Manual provides further information.

Message Handler



Process Environment

The Process Environment



Applications

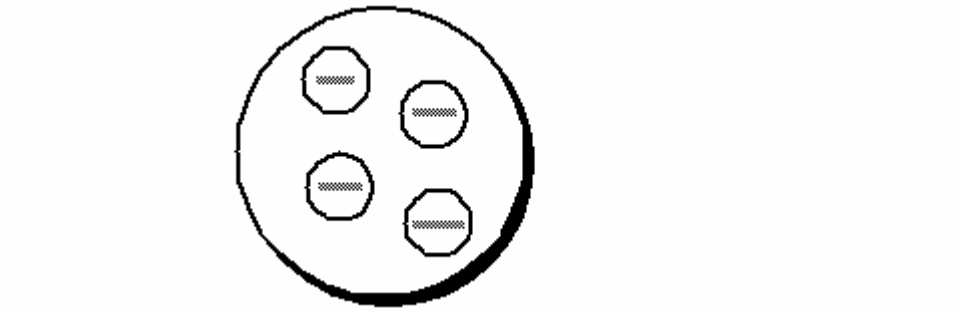
By applications we mean different user programs, which are not PLC programs. It can, for instance, be optimization programs or supervision programs. The application programs are written in the languages c, c++ or java. Also control programs can be written as applications.

PLC

The PLC job handles the execution of PLC code in the process environment. Each process environment has a base frequency, which is defined by the fastest PLCprogram in the node. The highest possible base frequency is 50 Hz.

All PLC code for one node is executed in one process. The PLC programs, which execute with the base frequency, execute in a thread. A thread is created for each frequency, which differs from the base frequency. Each thread has a priority. Usually,
the shortest cycle time has the highest priority, which means that a program that executes with high frequency cannot be blocked by a program with a lower frequency.

PLC process with three threads



The plc shell creates the subprocesses and starts them, when they are ready.

Any thread can read or write signals to the IO-cards. The read is performed before the executions of the plc-code each scan, and the write is performed after the execution.

In figure 'PLC process with three threads' the base frequency thread has a basic period of 20 ms. Furthermore you have two threads with periods of 40 and 120 ms. respectively.

Operator Environment

The operator environment handles the operator dialog.

The Operator Dialog

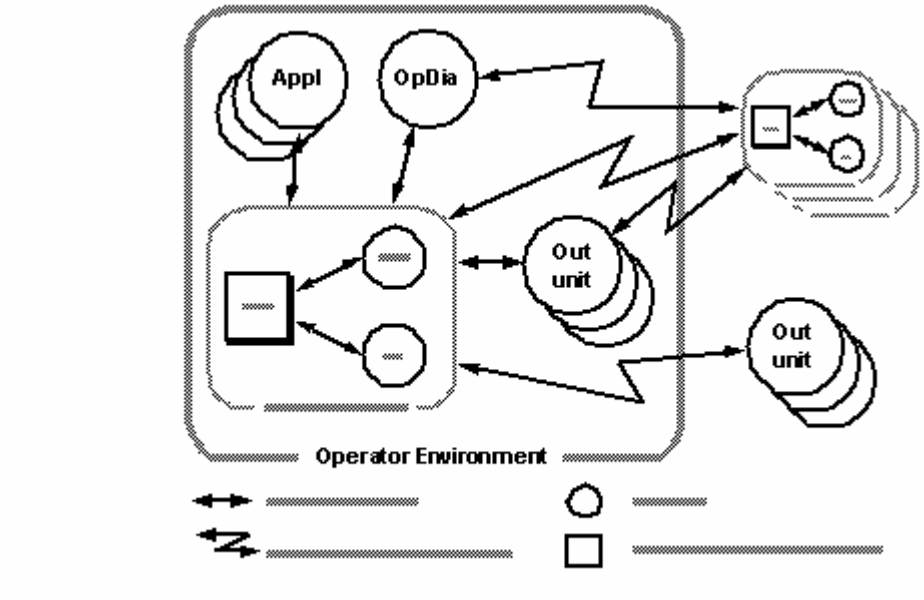
The Operator dialog handles the dialog between operator and process, and it consists of plant graphics, graphs, alarms, object displays, etc.

Message Out Unit

An out unit is a printer, a file, or an Operator dialog. Each out unit is represented by an object. The out unit executes its own process and connects to the Message Handler in the individual nodes. The out unit selects from which object hierarchies it wants to receive events. Today the following out units are available:

- Operator dialog, represented by the User object.
- Printer, represented by the EventPrinter object.

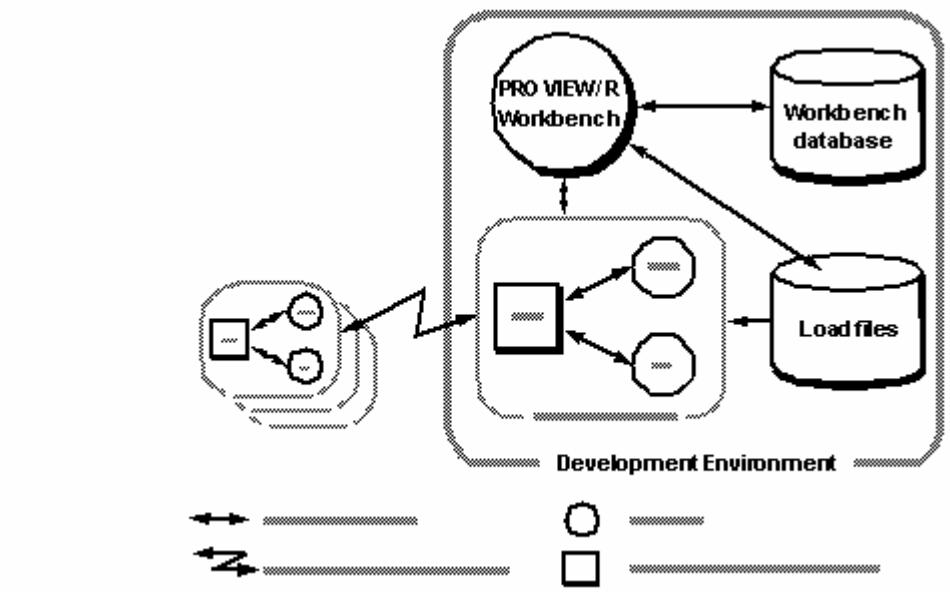
The Operator Environment



Developer Environment

The developer environment is used to configure, test, and maintain PROVIEW/R systems. The PROVIEW/R Workbench executes in this environment.

The Developer Environment



Workbench Database

All information on a PROVIEW/R system is stored in databases. When you are satisfied with the modifications you have made, you save them to the Workbench database.

Load Files

To start the PROVIEW/R Runtime Environment, a number of load files has to be created. These load files contain information on all objects and PLC programs in the system.

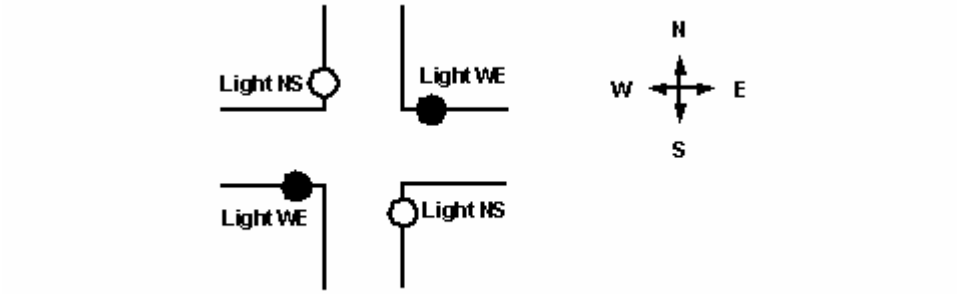
A Case Study

In this chapter we will give you an idea of how a PROVIEW/R system is created. The process to control in this case study is very simple - an intersection with four traffic lights - but it will give you an idea of the steps you have to go through when creating a PROVIEW/R system.

The traffic lights should be able to operate in two different operating modes:

- Normal: The traffic lights run normal cycle of red, yellow and green.
- Flash: The traffic lights are flashing yellow.

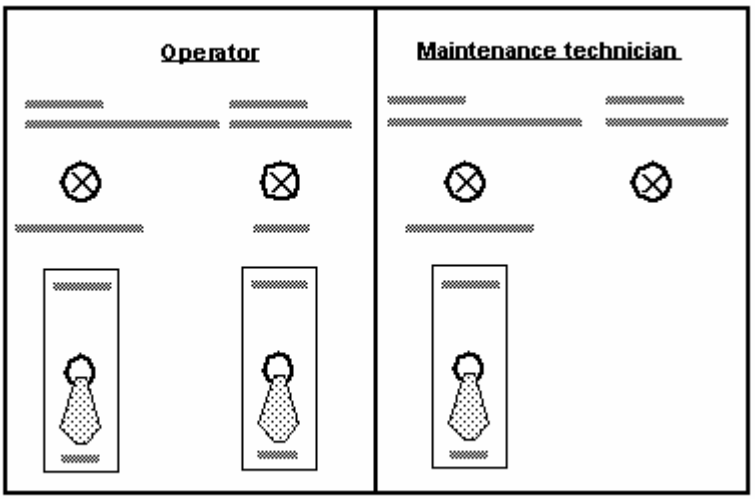
Traffic Lights in an Intersection



The operating mode of the traffic lights is decided by an operator at via operator station, or by a maintenance technician, who influences a switch. The maintenance technician can change mode, only if the operator have switched the traffic lights to service mode.

Figure *Traffic Lights, Control Panels* shows the different switches and indicators needed by the operator and maintenance technician respectively, to be able to monitor and control the system. These could be realized with plant graphics on the operator station or with hardware.

Traffic Lights, Control Panels



Specification of I/O

We start with analysing the task to decide what hardware to use.

Digital Outputs

We have four traffic lights but they can be connected in parallel, which means that we can treat them as two lights.

Three outputs per light: 2 × 3 = 6
Indication: Operating mode 1
Indication: Control 1

Total number of digital outputs: 8

Digital Inputs

The only digital input needed is for the maintenance technician's switch. The operator controls the process from the computer display, and this requires no physical signals.

Switch: Operating mode 1

Total number of digital inputs: 1

Analog I/O

No analog in- or outputs are needed for this task.

Specification of the Process Station

When we have decided upon the I/O needed, we can choose the hardware. We choose:

1 Linux PC with rack.
1 card with 16 digital inputs.
1 card with 16 digital outputs.

Specification of the Operator Station

1 Linux PC.

Specification of Plant Graphics

We need a display from which the operator can control and survey the traffic lights.

Administration

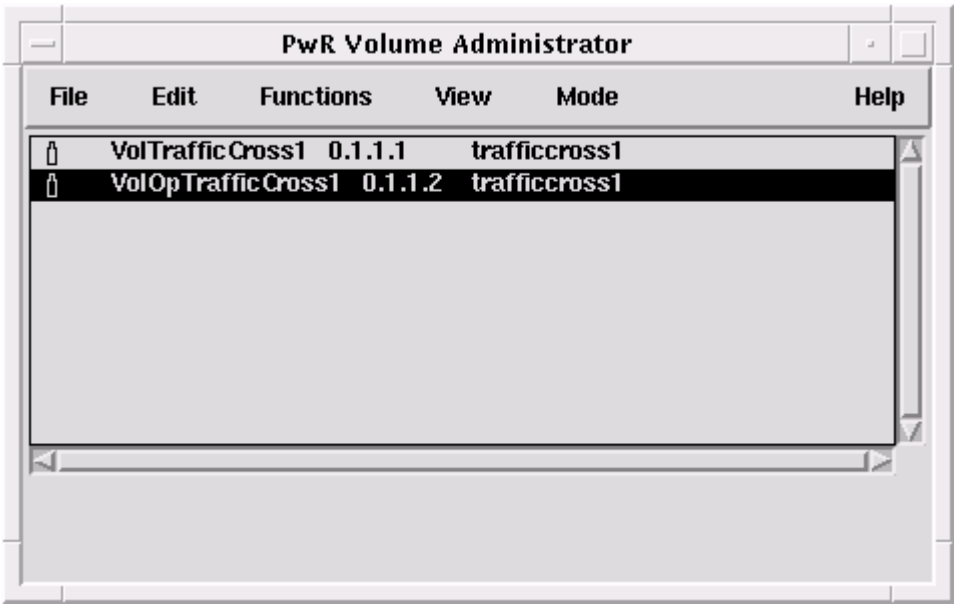
First we have to register a new volume, create a project, and, if necessary, create new users. For this we need

- A name for the project. We call it trafficcross1.
- Two volumes, one for the process station and one for the operator station.
- We need two users, one developer and one operator.

Volumes, projects and users are registred and created in the administrator tool.

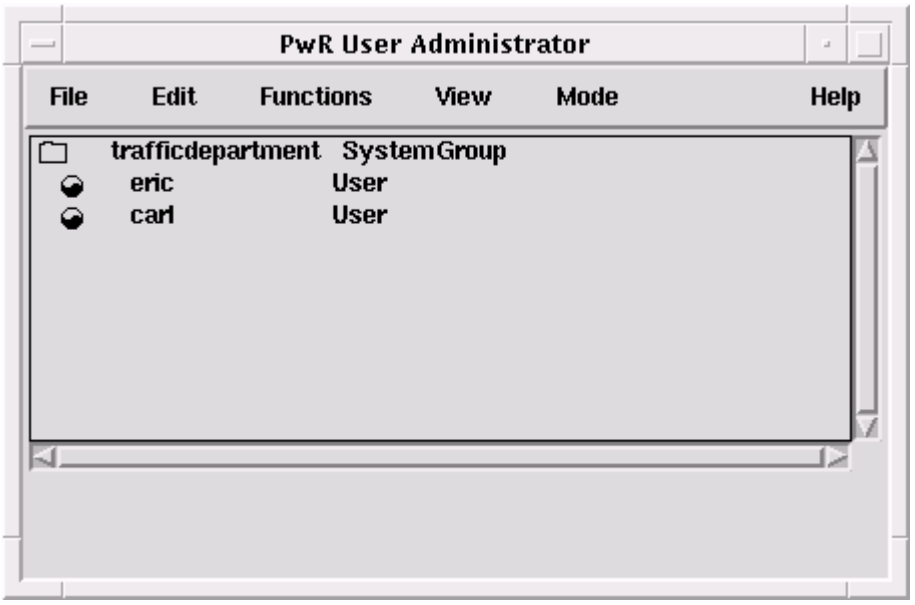
Register volume

For this project, we need two volumes, one for the process station, and one for the operator station. They are RootVolumes so we can choose an idle volume identity in the interval 0.1-254.1-254.1-254. We choose 0.1.1.1 for the process station and 0.1.1.2 for the operator station, and enter the volume mode in the administrator to register these volumes.



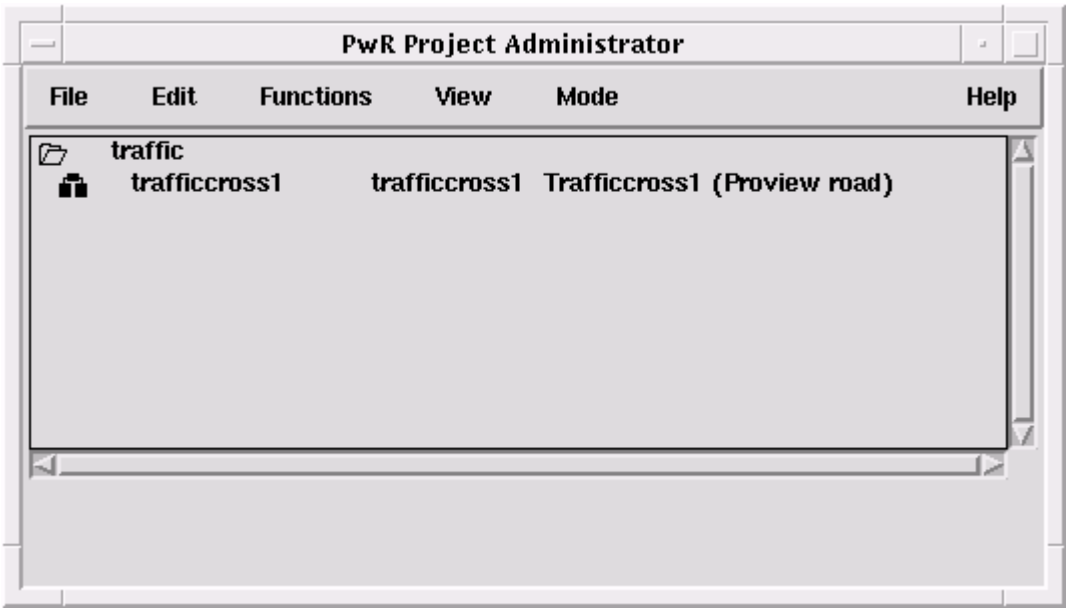
Create users

Eric is a developer at the traffic department, and Carl is an operator. They are both involved in all the projects at the traffic department, so we create a common systemgroup for all the projects and let them share users. We grant Eric developer and system privileges, and Carl operator1 privileges.



Create project

We create the project with the hierarchy name traffic-trafficcross1.



Configure the project

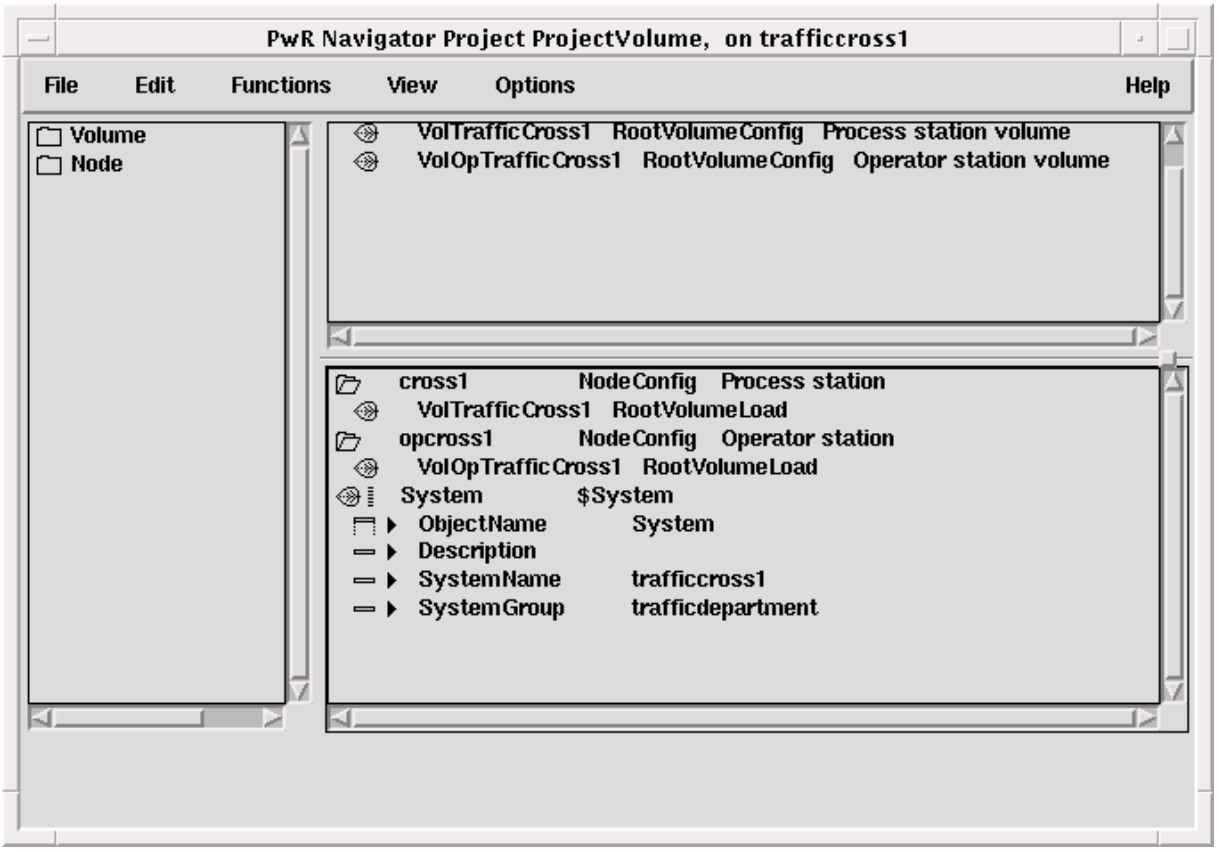
The project has a directory volume in which the nodes and volumes of the project are configured. In the upper window the volumes are configured with RootVolumeConfig objects. In the lower window the process and operator stations are configured with NodeConfig objects. The NodeConfig objects contains

- Nodename
- ip address of the node

- QCOM bus

Below each NodeConfig objects there is a RootVolumeLoad object which states the volume to load at runtime startup.

Note also the system object with the attribute SystemGroup that is assigned the value “trafficdepartment”. This grants the users eric and carl access to the project.



Plant Configuration

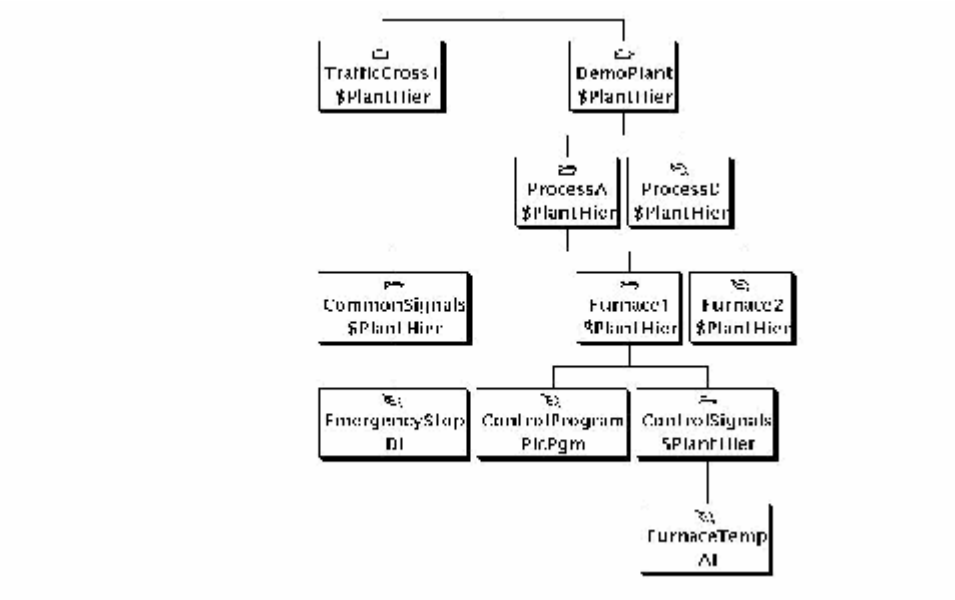
Once the task has been defined, next step is to configure the plant . The configuration is done in the Configuration Editor, see reference part, chapter The Configuration Editor page See The Configuration Editor. The plant is a logical description of the reality, which is to be controlled and supervised.

The plant is structured hierarchically. Examples of levels in the plant can be plant, process, process part, component, and signal. These are the logical signals represented by signal objects which will be connected to physical channels.

Sometimes it can be difficult to configure each signal in an initial stage, but it must at any rate be decided how possible signals shall be grouped.

Figure *An Example of a Plant Configuration* illustrates how a plant has been configured. We see how signals have been configured on different levels, and even how the PLC programs are configured in the plant.

An Example of a Plant Configuration



We choose to call our plant TrafficCross1 and we decide on the following structure:

- Two traffic lights, each one consisting of a green, a yellow, and a red lamp. Since the streets run north-south and west-east respectively, we call them TrafficLightNS and TrafficLightWE. Each lamp requires a signal. These are digital output signals and are called RedNS, RedWE, etc.
- A PLC program to control the traffic lights.
- A number of control signals to select operating mode and function. We choose to put them in one folder, ControlSignals. Table 2-1 shows the signals required.

Figure shows the resulting Plant Configuration.

Controlsignals for the Traffic Light

Signal Name	Signal Type	Function
ServiceSwitch	Digital input	A switch which the maintenance technician can influence to change the operating mode.
OperatorSwitch	Digital value	A value which the operator can influence to change the operating mode.
ServiceMode	Digital value	A value which the operator can influence to change the function to service mode.

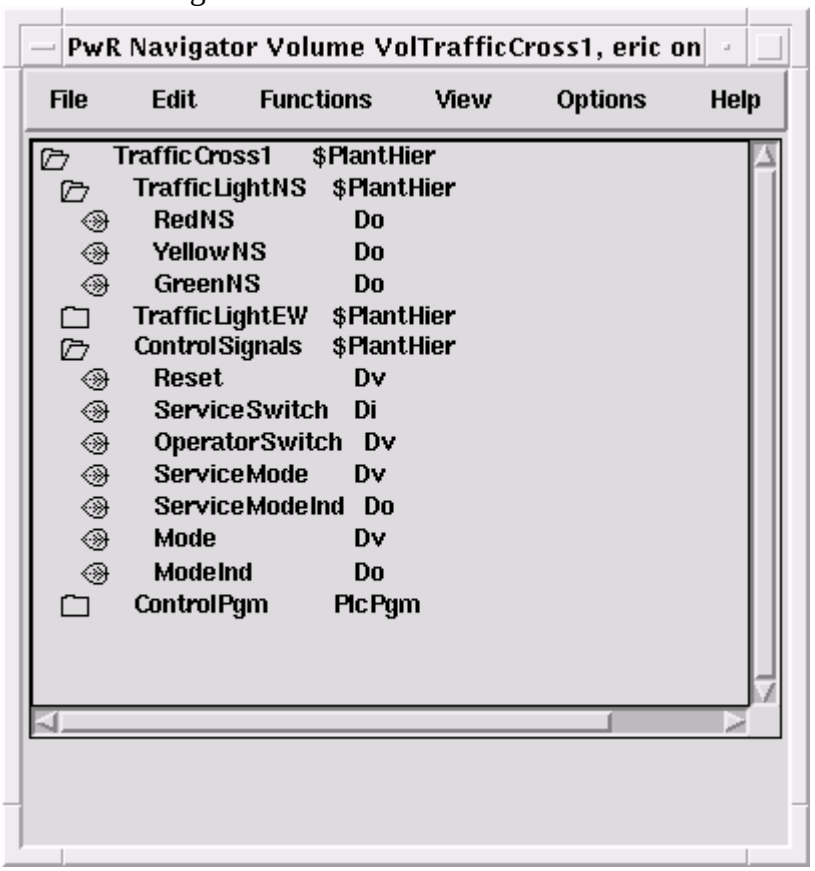
ServiceModeInd Digital output
A signal which shows the maintenance technician that the program is in service mode.

Mode Digital value
Indicates whether the program is in normal or flashing mode.

ModeInd Digital output
Indicates whether the program is in normal or flashing mode.

Reset Digital value
A value which is used to reset the program to initial mode.

The Plant Configuration of the Intersection



As you can see we have a plant object at the topmost level, TrafficCross1 of the class $PlantHier . We use other objects of class $PlantHier to group our objects. We also create an object, which defines a PLC program, the ControlPgm object of the class PlcPgm .

Node Configuration

When you have configured the plant, continue to configure the nodes. The configuration is done with the Configuration Editor, see reference part, chapter The Configuration Editor page See The Configuration Editor.

Process Station

In this example we choose to start configuring the process station. We name the process station RtpVax, because it is an RTP VX . It is advisable to give the process stations descriptive names.

In the analysis phase we decided that the process station should consist of the following hardware:

- 1 Linux pc
- 1 rack with 16 slots
- 1 card with 16 digital inputs (Di-channels)
- 1 card with 16 digital outputs (Do-channels)

The rack and the cards are configured in a manner much like what you would do physically. You have a node, place the rack in the node, the card in the rack, and the channels on each card.

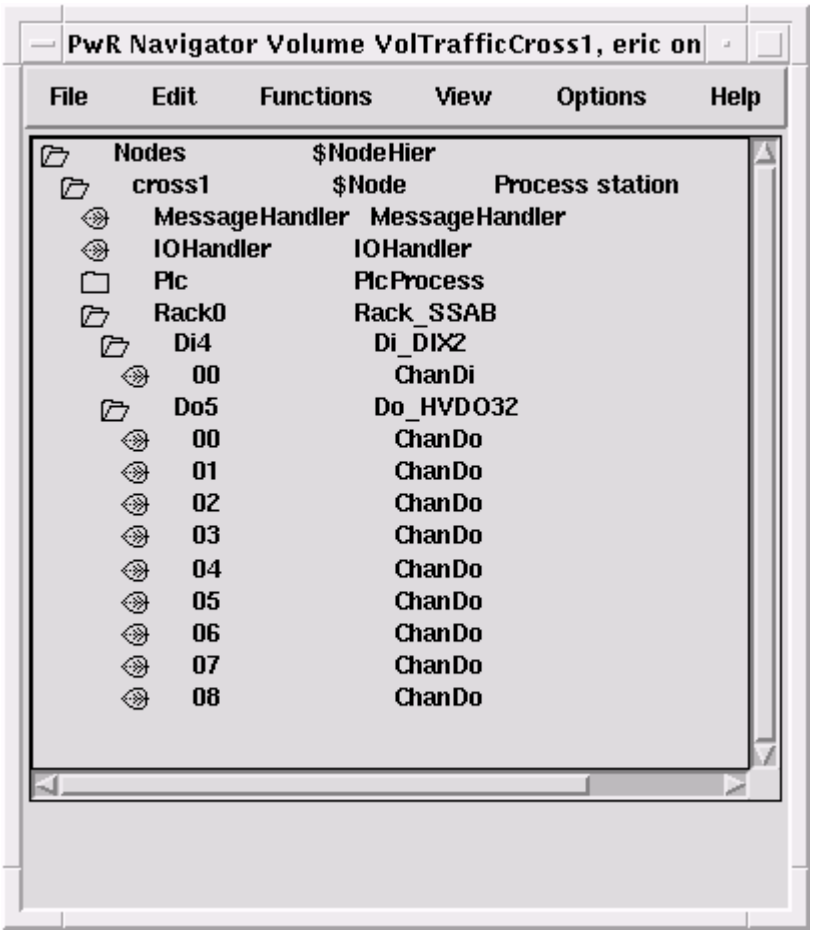
Each physical unit mentioned above is defined by an object class: the node object is of $Node class, you have different rack classes for different racks, a number of card classes for cards, and finally channel classes, e.g. ChanDi and ChanDo . A fast way to configure cards and rack is to use the function Configure Card in the Utility Window.

When you create the node object, some other objects are automatically created. These are the MessageHandler and IOHandler objects.

The MessageHandler object defines some important attributes which are used by the Message Handler on the node. The IOHandler object specifies the frequency at which analog signals will be read and written.

Figure *Configuration of the Process Station* shows the resulting Node Configuration.

Configuration of the Process Station



Each object has a number of attributes that you may have to change. For more information, see chapter 3, Configuring a PROVIEW/R System . To give an understanding of how to change attributes, some of the attributes in the node object of the process station are shown in figure See Attributes of the Process Station Node Object ($Node object). The attributes are edited with the Attribute Editor.

Note! The objects in this example are grouped under a node hierachy named "Nodes". Their full names will thus start with this name, as shown in the window title in figure See Attributes of the Process Station Node Object. See also figure 2-10, where the complete hierarchy is shown.

Operator Station

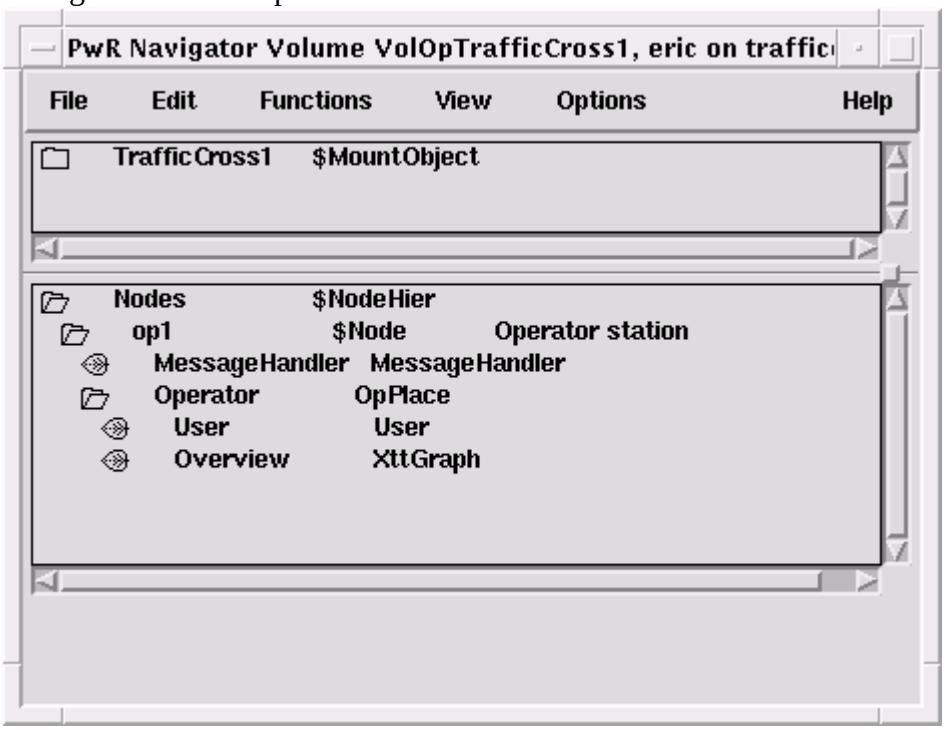
The plant hierarchy of the operator station is very simple. It only contains a MountObject that mounts the plant hierarchy of the process station. In this way, the objects of the process station will be known by the operator station.

The node hierarchy, the objects to configure the operator resides:

- The OpPlace object, which defines an operator place
- The User object, which defines a user

We also need a XttGraph object. This object contains the name of the Ge graph defining the plant graphics showing the road crossing. Figure *Configuration of the Operator* Station shows the configuration of the operator station.

Configuration of the Operator Station



Developer Station

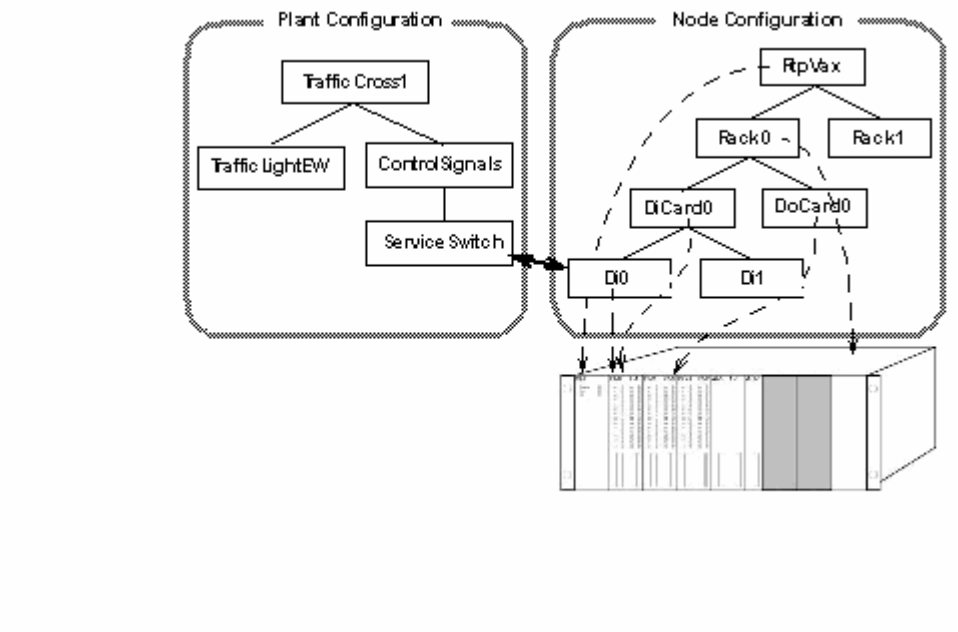
The developer station need not be configured, though in many cases, it is convenient to simulate the project in the developer station. This can be done by configuring the developer station in the project volume with another QCOM bus.

Connecting Signals to Channels

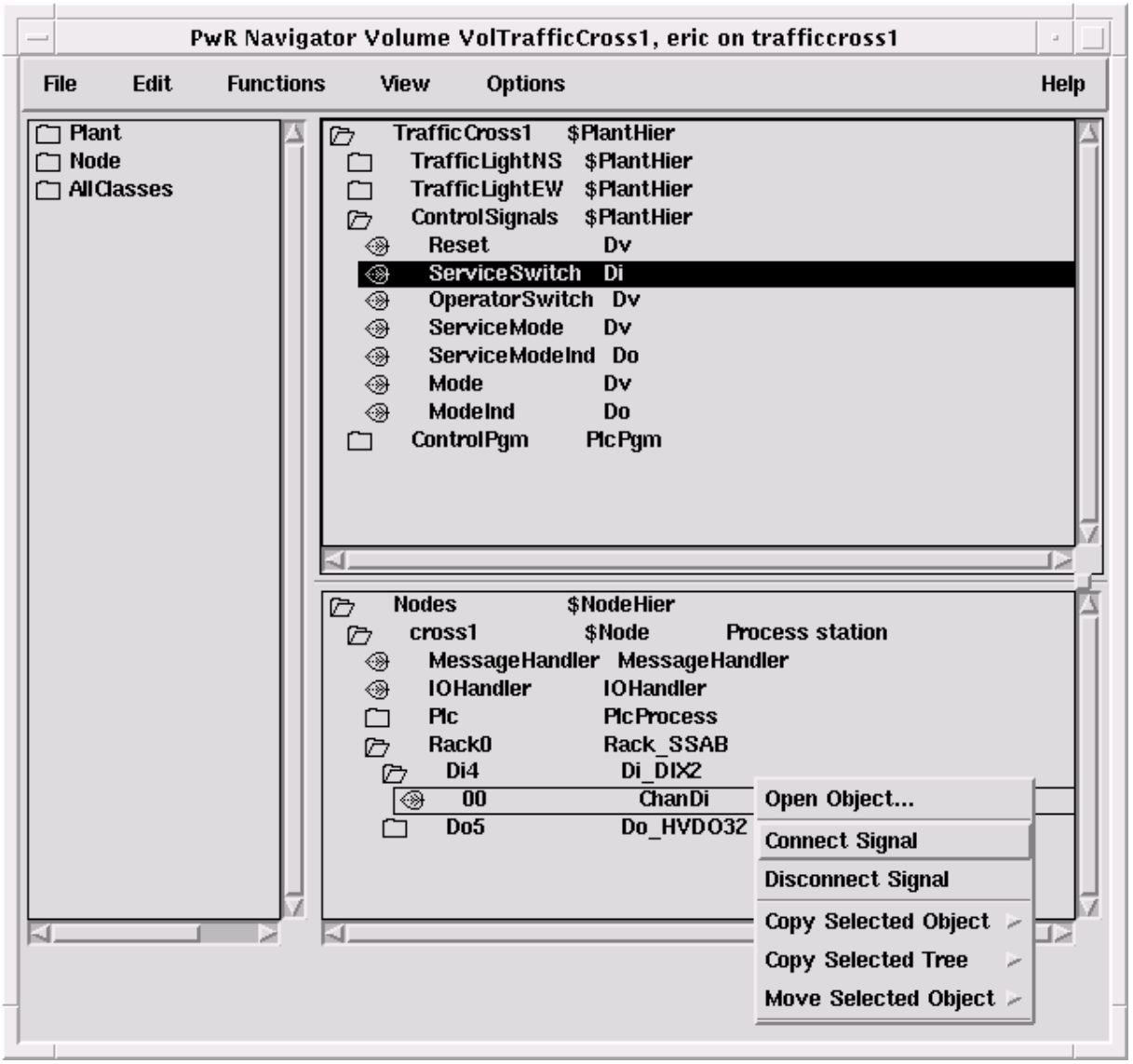
When you have configured the plant and the nodes, it is time to connect the logical signals to the physical channels. Each logical signal in the Plant Configuration must be connected to one channel in the Node Configuration (figure); a Di to a ChanDi , a Do to a ChanDo , an Ai to a ChanAi and an Ao to a ChanAo , etc.

Figure 2-10 shows how the digital signal ServiceSwitch is connected to the physical digital input channel Di0, using the pop-up menu. The steps to perform this are described in the chapter *Connecting Signals to Channels*.

Signal to Channel Connection



Connecting a Signal to a Channel



The figure shows how the digital signal ServiceSwitch is connected to the physical digital input channel Di4-00. The steps to perform this are described in the chapter *Connecting Signals to Channels*.

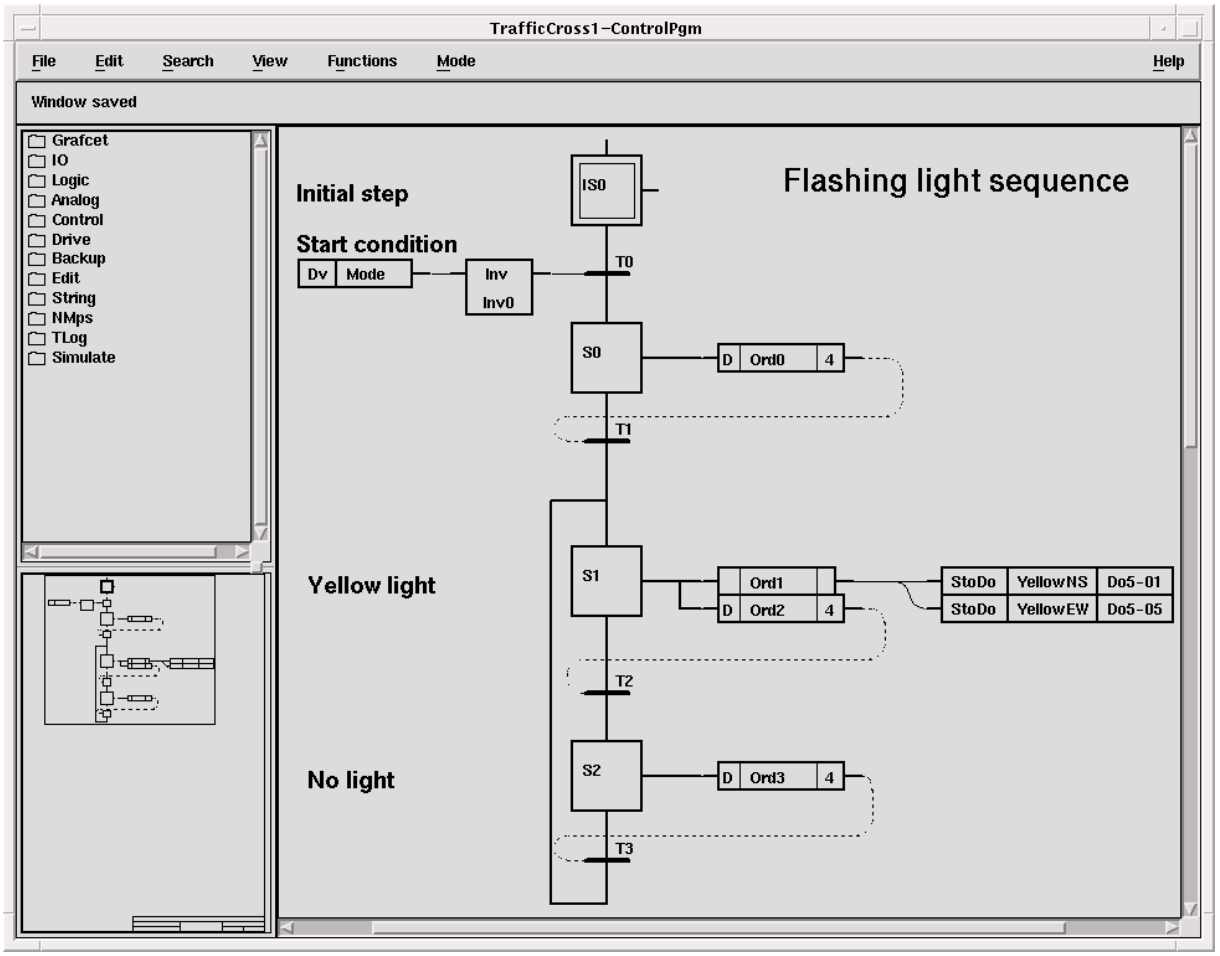
PLC Program

We use the Graphical PLC Editor to create PLC programs.

There is one attribute in the PlcPgm object must be stated:

- PlcThread, the plc thread object that should execute the PlcPgm.

The Graphical PLC Editor



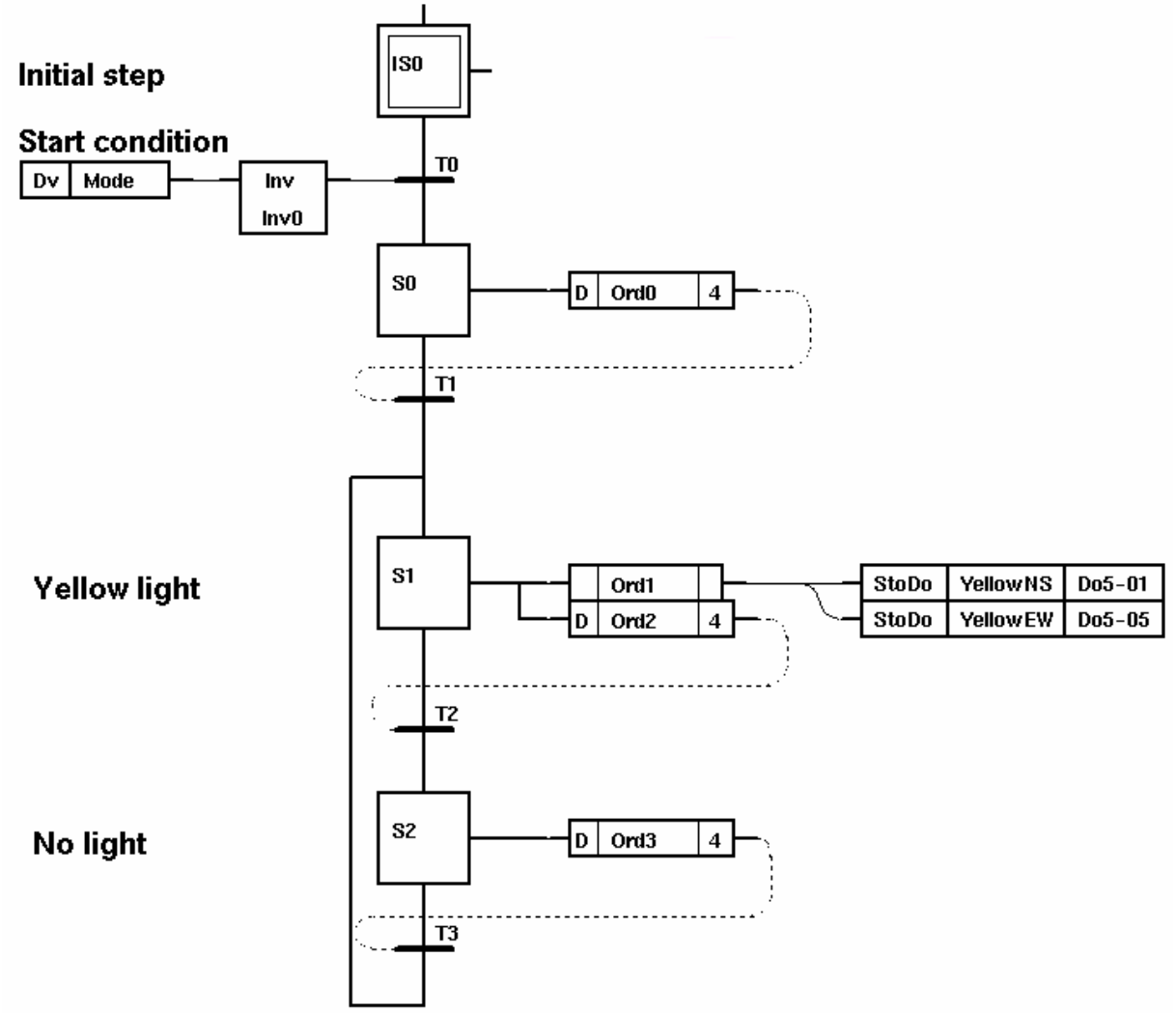
We will use the PLC Editor to create a sequential control program for the traffic lights. There are two ways to solve the problem concerning the two operating modes for a traffic light: normal and flash:

- Use one Grafcet sequence with conditional branches, i.e. one branch for the normal operating mode sequence and one for the flash operating mode sequence.
- Use two separate Grafcet sequences with different start conditions.

Here we choose to use the second alternative. In chapter 4, Graphical PLC Programming a more detailed description of Grafcet and sequential control can be found.

Grafcet programs are based on activating and deactivating a number of steps in sequence. In linear sequences only one step at a time can be active. To each step you tie a number of orders that are to be executed when the step is active. This can be e.g. to set (with a StoDo object) a digital output signal, which turns on a lamp. The PLC programs thus control the logical signals.

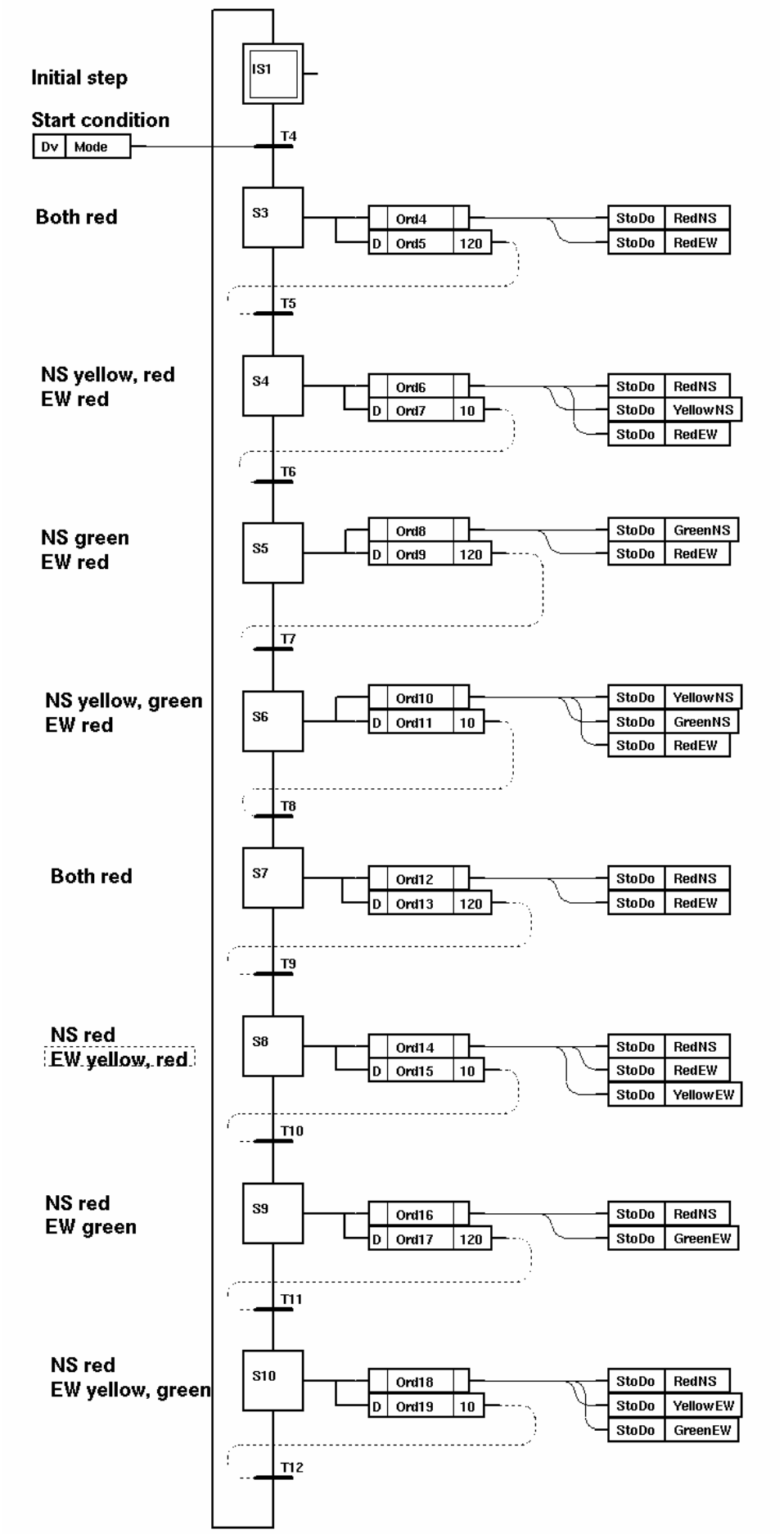
The Flashing Light Sequence



This is the sequence that will be executed when you want the lights to flash yellow.

The start condition for this sequence is inverted in relation to the start condition for the normal operating mode sequence. This implies that the two sequences cannot execute at the same time.

The Normal Sequence

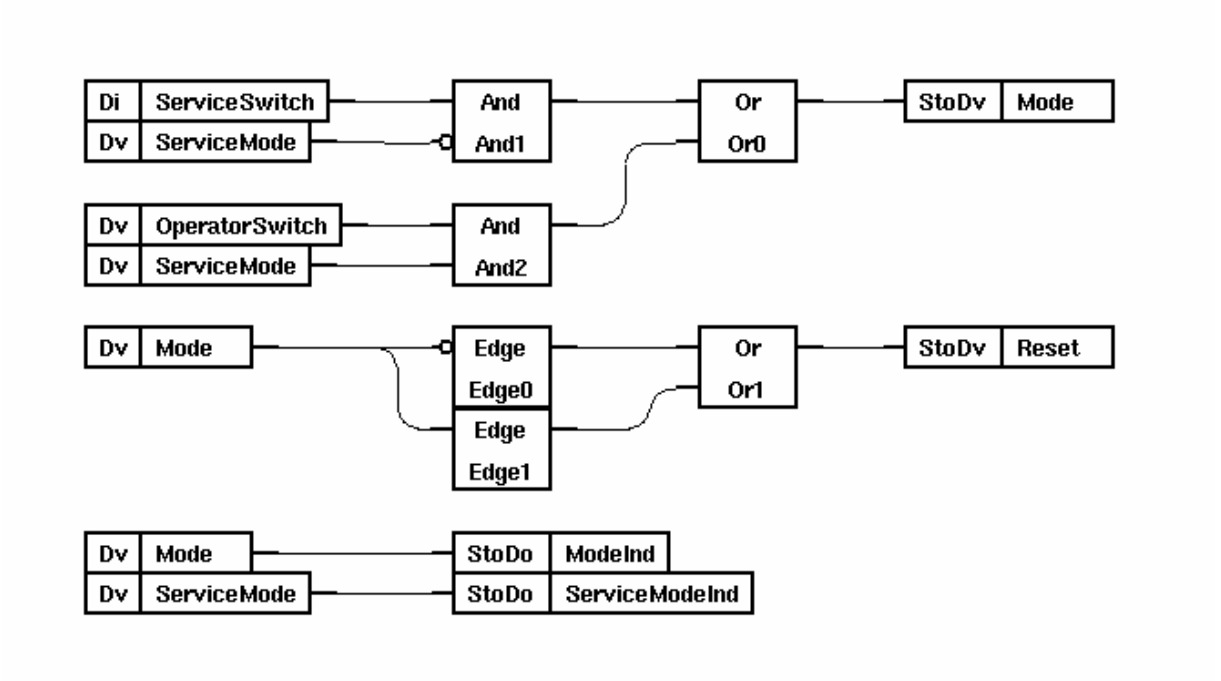

Initial step
IS1
Start condition
Dv
Mode
T4
Both red
S3
Ord4
D
Ord5
120
StoDo
RedNS
StoDo
RedEW
T5
NS yellow, red
EW red
S4
Ord6
D
Ord7
10
StoDo
RedNS
StoDo
YellowNS
StoDo
RedEW
T6
NS green
EW red
S5
Ord8
D
Ord9
120
StoDo
GreenNS
StoDo
RedEW
T7
NS yellow, green
EW red
S6
Ord10
D
Ord11
10
StoDo
YellowNS
StoDo
GreenNS
StoDo
RedEW
T8
Both red
S7
Ord12
D
Ord13
120
StoDo
RedNS
StoDo
RedEW
T9
NS red
EW yellow, red
S8
Ord14
D
Ord15
10
StoDo
RedNS
StoDo
RedEW
StoDo
YellowEW
T10
NS red
EW green
S9
Ord16
D
Ord17
120
StoDo
RedNS
StoDo
GreenEW
T11
NS red
EW yellow, green
S10
Ord18
D
Ord19
10
StoDo
RedNS
StoDo
YellowEW
StoDo
GreenEW
T12

The program for the normal operating mode is based on a traffic light following the sequence:

1. Red North-South, Red West-East
2. Red North-South, Yellow North-South, Red West-East
3. Green North-South, Red West-East
4. Yellow North-South, Green North-South, Red West-East
5. Red North-South, Red West-East
6. Red North-South, Red West-East, Yellow West-East
7. Red North-South, Green West-East
8. Red North-South, Yellow West-East, Green West-East
9. Back to step 1

The program starts in the initial step. If the start condition is fulfilled, step S1 will become active and the red lamps are turned on. After a certain time, step S1 will become inactive and step S2 will become active, and a yellow lamp will also be turned on, and so on. When step S8 has been active for a certain time, it will be deactivated, and the initial step is once again activated.

Trigger Signals



The program above shows the logic that controls different operating modes.

At the very top to the right you set the Dv signal "Mode". If this is set to a logical 1, the sequence for the light's normal operating mode will be run, otherwise the sequence for flashing lights will be run.

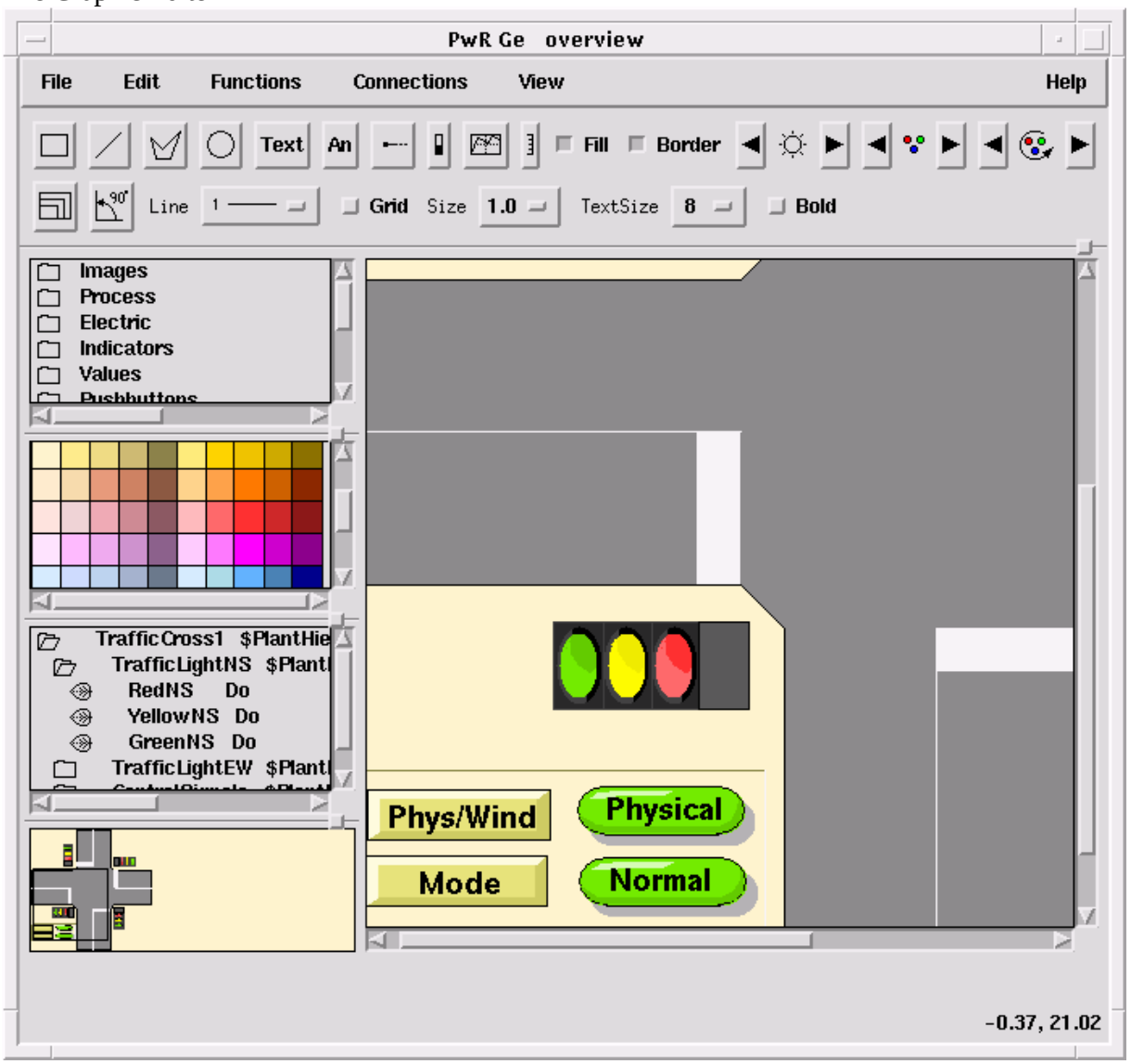
The Dv signal "Reset" will be set to a logical 1 during one execution cycle when the signal Mode changes value. This implies that the two Grafcet sequences will return to the initial step. The chosen sequence will be executed again when Reset is set to a logical 0.

The PLC programs you have created must be compiled before they can be executed on a process station.

Plant Graphics

Plant graphics are often used as an interface between the operator and the process. Plant graphics are created with the Graphic Editor (Ge).

The Graphic Editor



Plant graphics can contain dynamics, which are connected to the logical signals, e.g.:

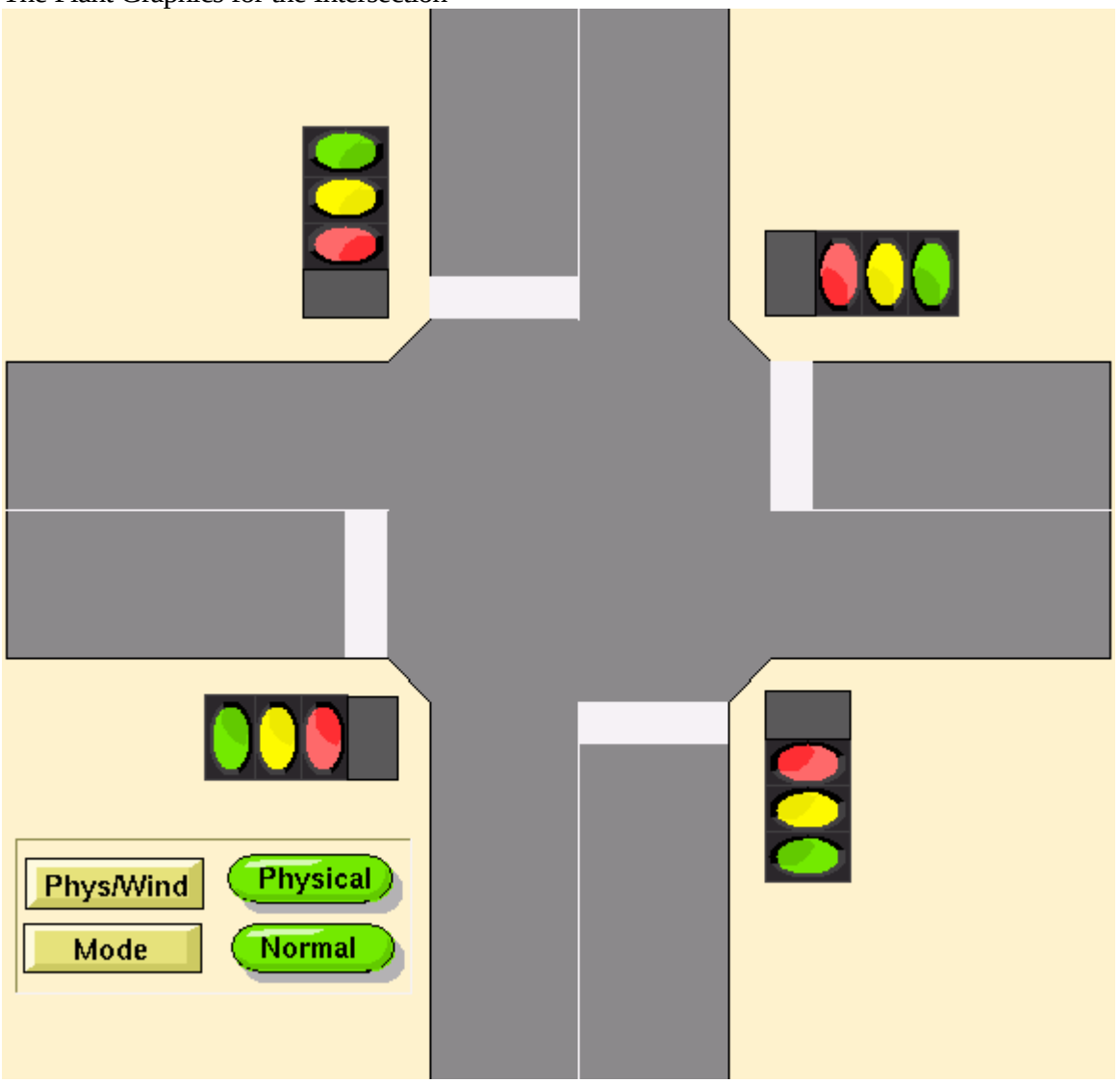
- Text that becomes visible when a signal reaches a certain value

- Graphical objects that change color when a signal reaches a certain value
- Graphical objects that become invisible when a signal reaches a certain value
- Graphical objects that move depending on the value of a signal

You can also place push buttons in the plant graphics, which the operator can use for changing values of digital signals. To change analog signals you use an input entry field.

In our example we choose to make a plant graphics, showing a road crossing, where the traffic lights (red, yellow, and green) are dynamic as shown in figure . How to create the plant graphics is described in chapter 5 Creating Plant Graphics .

The Plant Graphics for the Intersection



Load Files

Up to this point, we have configured and created programs in the Workbench. We now want to run the system.

Before starting the runtime environment we must create load files for each node in our system, and also a boot file. This is explained on page See Creating Load Files.

Create a project

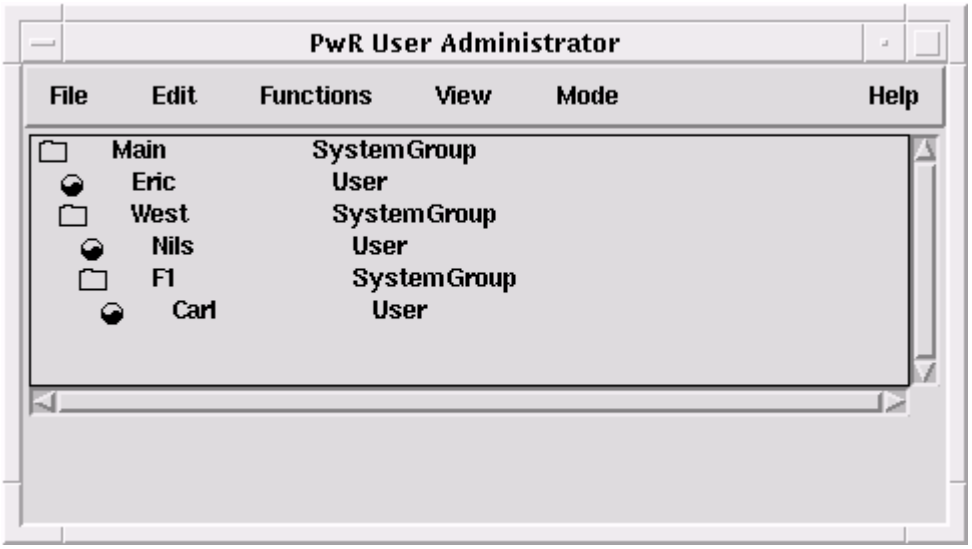
Add users

To gain access to the Proview development and runtime environment you need to login with username and password. Users are kept in the user database and are granted privileges which states the users authority to make changes in the system.

Systems that share the same users are grouped into a system group, and the the users for this group is defined. You can also build a hierarchy of system groups where child groups inherit the users of their parent, and additional users can be defined for each child.

A system is connected to a systemgroup by the SystemGroup attribute in the $System object. The notation for a system group in a hierarchy is the names of the group separated by a point, for example ‘Main.West.F1’.

In the example Eric is responsible for all the systems in the plant, and is defined on the highest level in the hierarchy. Nils is working with the west side of the plant and is defined on the ‘West’ system group. Finally, Carl working with the systems in the F1 part of the plant. All system groups has the attribute UserInherit.



Users and systemgroups are created in the administrator:

- Start the administrator with the command ‘pwra’.
- Enter user mode from the menu *File/Show Users*.
- Login by entering the login command. Open the login prompt from the menu *Functions/Command* and enter ‘login’ on the command line. If the systemgroup

'administrator' is present you also has to add username and password to a user defined in the administrator systemgroup.

- Enter edit mode from the menu *Mode/Edit*.
- Open the New System Group (*Edit/New SystemGroup*) dialog and enter name and attributes. Supply the full hierarchy name, e.g. 'Main.West'.
- Select a system group and open the *New User* dialog (*Edit/New User*) and enter username, password and privileges for the user.
- Click in the Ok button.
- Logout with the command 'logout'.

The user database reside in the directory $pwra_db.

Register volumes

All volumes in a network has to have a unique volumname and volume identity. To assure this, all volumes is registered in a global volume.

The registration is done by the administrator:

- Start the administrator with the command 'pwra'
- Enter volume mode from the menu *File/Show volumes*.
- Login
- Enter edit mode from the menu *Mode/Edit*.
- If a volume in the volume list is selected, the new volume will be inserted after this volume, otherwise it is inserted last in the list.
- Open the New volume dialog (*Edit/New Volume*), enter volume name, volume identity, and system name.
- Click on the Ok button.
- Logout with the command 'logout'.

Volume name

The name of the volume, max 32 characters.

Volume identity

The volume identity is a 32 bit word specified in the form v1.v2.v3.v4 where v1, v2, v3 and v4 are numbers in the interval 0-255. Dependent on the class of the volume, the numbers can be chosen in separate intervals.

RootVolumes	0. 1-254. 1-254. 1-254
User ClassVolumes	0. 0. 2-254. 1-254

The DirectoryVolume always has the identity 254.254.254.253

Create project

A project is a number of nodes and volumes that share the same development environment. Usually it consist of some process stations and a couple of operators stations that control a part of the plant, but there are no restrictions in the size of a project. You can choose to have each node in its own project or all the nodes in the same project.

- All the nodes in a project (on the same bus) has a nethandler link between each other.
- All the volumes and nodes share the same directory tree.
- All node has to be upgraded to new Proview versions at the same time.

A common size is 1 – 10 nodes in a project. Too many nodes will increase the communication overhead and make it harder to upgrade the project.

Create the project in the administrator:

- Start the administrator with the command 'pwra'.
- The project mode is default when starting the administrator. If the current mode is User of Volume, enter Project mode from the menu *File/Show projects.*
- Login.
- Enter edit mode from the menu *Mode/Edit*.
- Open the Create Volume dialog (*Edit/Create Project*), and enter project name, base version, path, hierarchy and description.

Project name

A project has a *project name* that identifies the project in the development environment. It is similar the *system name* that identifies the project in the runtime environment, but it doesn't have to be the same. Actually a system can have several projects in the development environment. When upgrading or making a major modification of the system, it is advisable to take a copy of the project and keep the currently running version of the system available for minor corrections. The copy is then created under a new project name, though it has the same system name.

Base

Proview is a multiversion system, i.e. different versions of proview can be installed in the same development environment and projects of different proview versions can coexist in the same development environment. A project points at a Proview base, e.g. V3.4, V4.0, and when creating a project you have to choose which base the project should point at.

Path

The project consist of a directory tree where databases, source files, archives etc is stored. The path is the root directory of this tree.

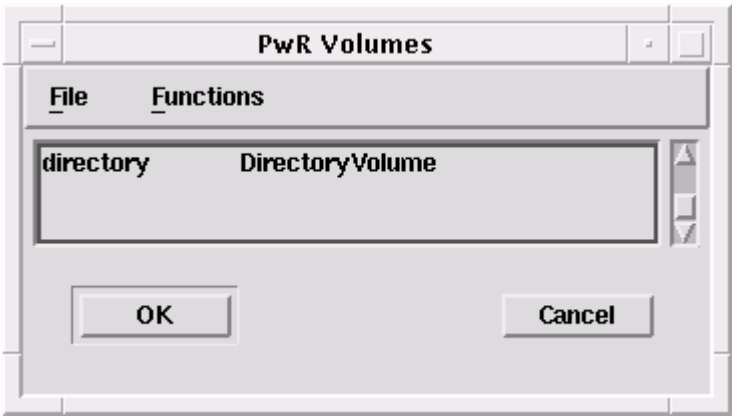
Hierarchy

The administrator can order the projects in a tree structure, and the hierarchy is the name of the nodes in the tree separated by ‘-‘. The hierarchy nodes of the tree will be created by the administrator to match the specified name.

The DirectoryVolume

When the project is created, it is found in the administrator project tree. You open a project by selecting it and activate *Edit/Open Project* in the menu. The workbench is now opened for the project, and the Volume selection window is displayed, showing all the volumes in the project. So far, only the DirectoryVolume is created, and our first task is to configure this volume, with the volumes and nodes of the system.

Select the DirectoryVolume and click on the Ok button to open the configuration editor for the volume.



The Configuration Editor

The configuration editor displays two windows, and for the DirectoryVolume, the upper shows the volume configuration, and the lower the node configuration.

RootVolumeConfig

First we create RootVolumeConfig objects for all rootvolumes in the project.

- Enter the edit mode from the menu Edit/Edit mode. Now the palette is visible to the right in the window, and maps can be opened with a click on the map symbol or a double click on the text.
- Open the Volume map and select the RootVolumeConfig class.
- Click with MB2 in the volume configuration window, and the object is created.
- Select the object and open the object editor from the menu Functions/Open Object...
- Select ObjectName and activate *Functions/Change value* in the object editor menu.
- Enter the name of the object. The name should be the same as the name of the volume.
- Close the object editor.

Create the RootVolumeConfig objects for the other rootvolumes of the project. For the following objects you can control the position of the object. If you click with MB2 on the object name of an object, the new object will be a sibling to this object. If you click on the leaf or map symbol, the object will be a child.

It is also possible to display the attributes of an object directly in the configuration editor:

- Press Shift and click MB1 on the object to open the object
- Select an attribute and activate *Functions/Change value* to modify a value.

When the session is saved the volumes and their databases will be created.

NodeConfig

Next step is to configure the nodes of the project in the node configuration window. Create a NodeConfig object for each process and operator station. When the NodeConfig object is created, some additional child objects are also created

- a RootVolumeLoad objects that states which root volume to load at runtime startup for this node. Set the name of the object to the name of the RootVolume.
- a NodeHierarchy objects for objects that controls the distribution of files from the development environment to the process and operator stations.

System object

Create also a $System object in the node configuration window. The system object has ths attributes SystemName and SystemGroup.

- The system name in this state, often is equal to the project name.
- The system group attribute makes the system a member of a system group in the user database, which defines the users for the system. Once the system object is created you have to state a valid username and password when entering the workbench.

Save the session from the menu *File/Save*. If the configuration passes the syntax check, you will receive a question if you want to create the configured volumes. Answer Ok to these questions.

If the volume selection window is opened now, *File/Open...* in the menu, all the configured volumes are displayed. The next step is to configure a RootVolume.

The RootVolume

A root volume is opened from the volume selection window. Select the volume and click on the Ok button. This will start the configration editor for the root volume. As for the DirectoryVolume it is separated in two windows, but this time, the upper window shows the plant configuration and the lower the node configuration.

Plant Configuration

The Plant Configuration describes the different plants which you can find in the PROVIEW/R system. A plant is a logical description of e.g. a production process, functions, equipment, that is to be controlled, supervised, etc. An example of a Plant Configuration is shown in figure The Plant Configuration of the Intersection.

$PlantHier Object

The top object in the plant hierarchy is the $PlantHier object. This object identifies the plant or parts of it.

The $PlantHier object is used to group objects and to structure the plant. This object can, for instance, be used to group signal objects.

Signal Objects

The signal objects define logical signals, or points, representing a quantity or value somewhere in the process; whereas contrast to the channel objects, which define physical signals. The signal objects are generic, i.e. they can be used with any I/O-system.

There are two classes of signals that cannot be connected to hardware signal, i.e. the Av and Dv objects (Analog value and Digital value respectively). These two objects are used to store a real number respectively logical value.

The actual value of the signal is defined by the attribute ActualValue .

At present the following signal objects are available:

Ai Is used to describe analog input signals.
Ao Is used to describe analog output signals.
Av Is used to describe real numbers.
Di Is used to describe digital input signals.
Do Is used to describe digital output signals.
Dv Is used to describe logical values.
Co Is used to describe counter signals.

Note! The PLC program can read signals placed on remote nodes, but cannot write to them. This is due to the fact that the PLC program sets up subscriptions for all signal objects, and if the object is not placed in the same node as the PLC program, you write in a copy of the object. See chapter on subscriptions on page ...

PlcPgm Object

The PlcPgm object defines a PLC program. It is possible to have several PLC programs in a plant. The following attribute must be given a value:

- ThreadObject indicates the plc thread where the program is executed. It references a PlcThread object in the node configuration.
- If the program contains a GRAFCET sequence, the ResetObject must be given. This is a Dv, Di or Do that reset the sequence to its initial state.

Backup Object

The Backup object is used to point out the object or attribute, for which the backup will be made. It is also indicated whether storing will take place with fast or slow cycle time.

MountObject

The MountObject mounts an object in another volume. The attribute Object specifies the mounted object.

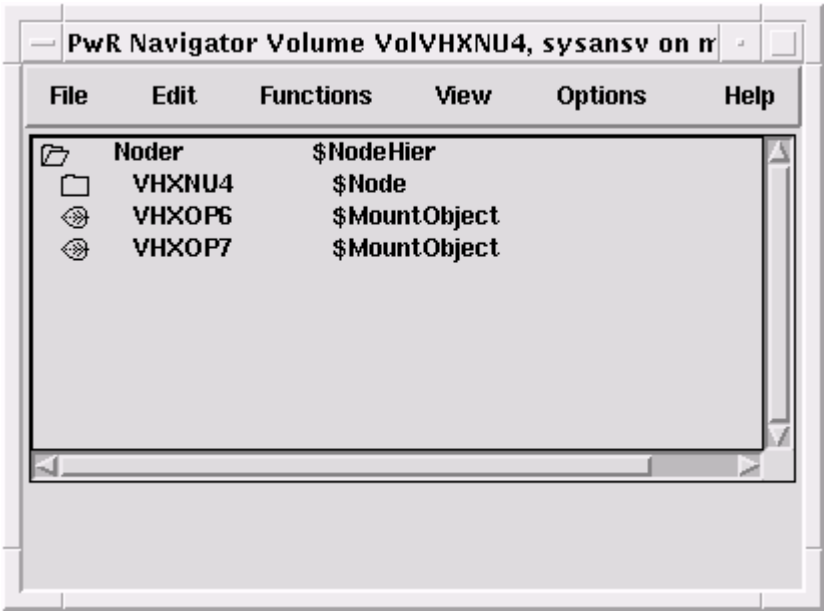
Sv Object

The Sv (StringValue) object is a string of maximum 79 characters. The string can, for example, be used to display a text in a process graph.

Node Configuration

The Node Configuration defines the nodes of your PROVIEW/R system. The nodes are named and their contents specified.

Node Configuration



Node Hierarchy Object

The node hierarchy object is used to group objects in the Node Configuration. This object is of the $NodeHier class. The object can be used to group for instance $Node objects or Graph objects.

$Node Object

To define the nodes of the system, you use node objects. The node object is of the $Node class. When the node object is created, a number of server and operator objects are

Node Configuration of a Process Station

Below follows a description of how to configure a process station.

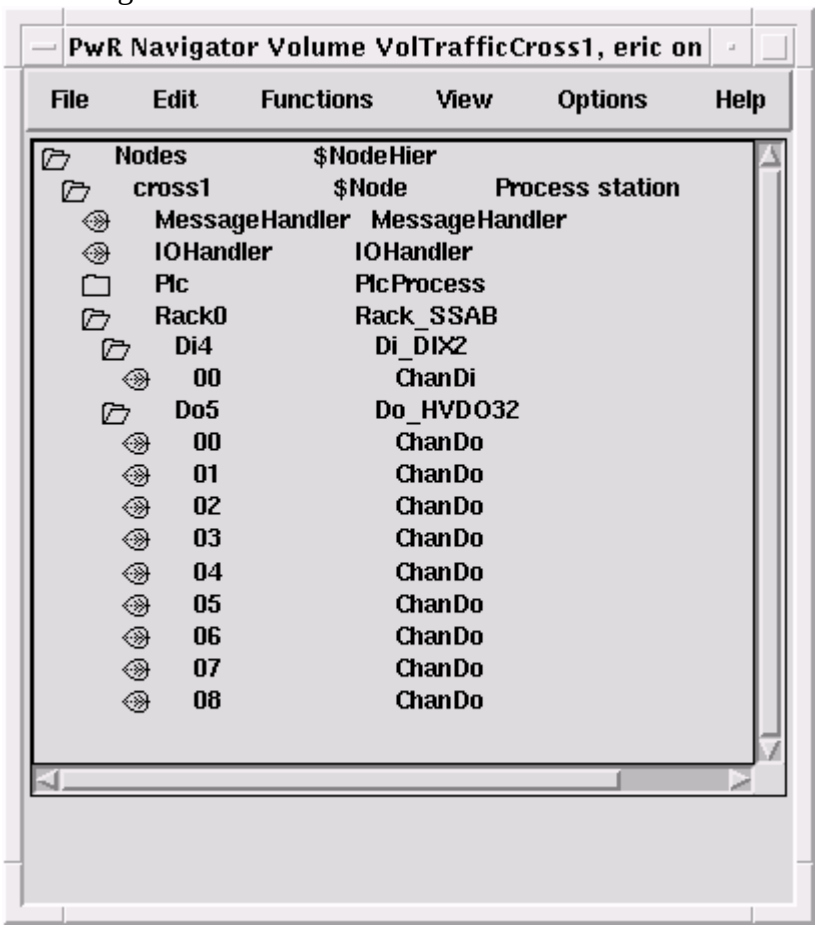
I/O Objects

When you configure the I/O of the process station, you start by creating a rack object. This object will later become the parent of a number of card objects, which in their turn become parents to a number of channel objects (see figure).

As far as attributes for the channel objects are concerned, please study PROVIEW/R Objects Reference Manual . For information about racks and cards, see PROVIEW/R RTP I/O-system Reference Manual .

By using the Utility Window you can also very easily configure a card including its channels.

I/O Configuration



Acquisition Objects

If you want to store attribute values, you use the acquisition objects. The values are stored in a ring buffer. You can decide how many samples the object will contain, as well as sample time.

Two types of acquisition objects are available:

DsTrend is used when you want to continuously store slow progresses. Is used for sampling times of 1 second and more.
DsFast is used when you want to store fast progresses. You can choose to store data continuously or a number of samples after activating a trigger signal. The highest storing frequency is 50 Hz.
Note! Place the acquisition object and the attribute to be stored in the same node!

See PROVIEW/R Objects Reference Manual for additional information.

Trend Graphs

To show trend graphs, which have been collected by means of acquisition objects, you use a PlotGroup object.

A PlotGroup object can handle 4 trend graphs (DsTrend object), or 4 fast graphs (DsFast object). The graphs in a PlotGroup must have the same base frequency. It is therefore suitable to let each plot group handle DsTrend or DsFast objects from one node only.

The operator can access the PlotGroup objects, which are placed in the same node as the operator place object.

You find more information on trend graphs in PROVIEW/R Objects Reference Manual and in PROVIEW/R Operator's Guide.

MessageHandler Object

On each station in a PROVIEW/R system executes a Message Handler. This process handles the supervisory objects (DSup and ASup objects). When an event has been detected, a message is sent to the out units, asking for that particular event.

The MessageHandler object specifies important attributes for the Message Handler, including e.g. how many messages that should be buffered in the node. The object is created automatically when you create a node object. The object created as a child of the $Node object.

IOHandler Object

Analog signals is read with a frequency specified by the IOHandler object. This object is created automatically when you create a node object. The object created as a child of the $Node object.

ReadWriteFlag specifies whether to address physical hardware or not. It is used, for instance, if you want to test your program without influencing the physical hardware.
0 = Does not address the physical hardware.
1 = Addresses the physical hardware. (Default)
SimulFlag indicates whether to use the hardware or not.
0 = The hardware is used. (Default)
1 = The hardware is not used.
It is recommended to study the PROVIEW/R Objects Reference Manual on the $Node object.

Trend graphs
...

Backup_Conf Object - Configuration Object for Backup

Sometimes it may be desirable to have a backup of a number of objects in your system. In that case you place a backup configuration object, Backup_Conf , under the node object. The

backup is carried out with two different cycles, one fast cycle and one slow. The disk with the backup file can be located in the same node or somewhere else in the network.

If you use a backup file, you will at re-booting of PROVIEW/R be asked on the console of the station, whether you want to read in the backup file. If you do not reply to the question within 30 seconds, the backup file will automatically be read and the object data will be re-created.

In order to indicate which objects/attributes that should be backed up you use backup objects. How to configure a Backup object is described on page See Backup Object.

Node Configuration of an Operator Station

Below follows a description of how to configure an operator station. One or more operator places are tied to the operator station.

Operator Place Object

To define an operator place you place an object of the OpPlace class under the $Node object. If you want to use an X terminal as an operator place, you create an OpPlace object for each X terminal that the node shall handle.

The following attributes must be given values:

- OpNumber indicates the number of the operator place. Every operator place in the PROVIEW/R system should be allocated a unique integer.

User Object

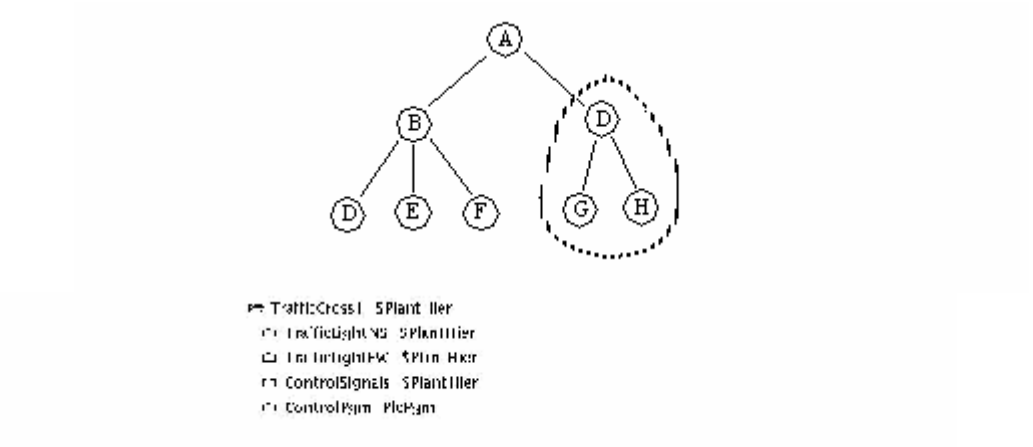
For each operator place you must define an out unit of the user type (operator). To do this you use a User object. You can only have one user per operator place.

The following attributes must be given values:

- UserName indicates the name of the user.
- OpNumber must have the same number as the operator place.
- MaxNoOfEvents indicates the number of events which the operator's event list can hold at the same time. We recommend an initial value of 200. This can be increased when required.
- SelectList indicates the object hierarchies in the Plant Configuration, from which the operator will be receiving events.

If we look at the figure above, which illustrates the plant A, and assume that we want to receive events only from the encircled objects, we state A - C as an alternative in the select list. This choice means that we will be receiving events from the supervised object C, and from all supervised objects, which have C as their parent.
Another example:

We look at the figure below, which illustrates the plant TrafficCross1. If you want to receive all events from the plant TrafficCross1, you state TrafficCross1 as an alternative. TrafficCross1 handles two traffic lights, TrafficLightNS and TrafficLightWE. Let us say that we want events only from TrafficLightNS. In that case we state TrafficCross1-TrafficLightNS instead of TrafficCross1.



If you want to receive messages from the Watchdog object of a node, you must also state the hierarchy name of the node, for instance RtpVax.

In FastAvail you specify the complete hierarchy name of the Graph object, which will be possible to start from the graphics buttons of the Operator Window (see figure See The Operator Window), for instance Demo-Traff
NoFastAvail specifies the number of graphics buttons to be used. You can have 0- 15 push buttons. Buttons which are not used become invisible. The number should be in accordance with the number of graph objects in FastAvail.

The Operator Window
...

The Plant Graphics Object - the XttGraph Object

In order to be able to show plant graphics which are unique to the project in the operator station, you must configure XttGraph objects. These objects define, for instance, what the files with the plant graphics are called.

These are the objects, to which the User object attribute FastAvail refers.

In the XttGraph object the following attribute must be filled in:

- Action

Acquisition Objects

If you want to store analog attributes which are owned by the operator station you must configure acquisition objects, see page See Acquisition Objects.

Signal to Channel Cross Connection

The last thing to do when configuring your system is to connect signals in the Plant Configuration with channels in the Node Configuration.

Di signals should be connected to Di channels, Ao signals to Ao channels, etc.

This is described in the chapter: Connecting Signals to Channels on page See Connecting Signals to Channels.

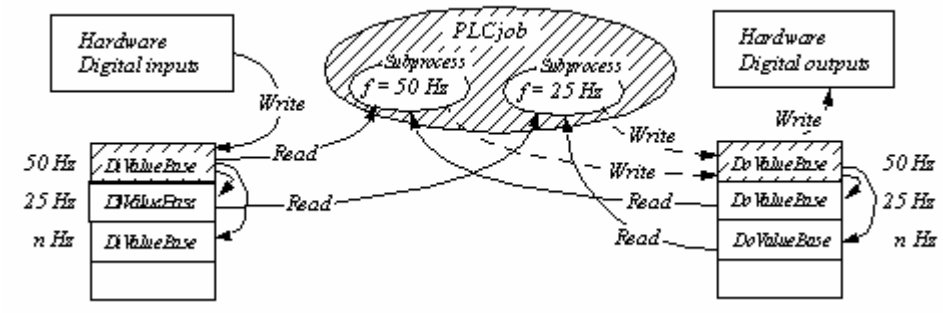
The Area Objects - I/O Copying Areas

I/O copying implies that all signal values are read at the beginning of the execution of the PLC thread, which means that even if a signal value is changed during the execution of the PLC process, the original signal value is maintained during the entire cycle. Threads with different frequencies read from different areas but write to one area, as shown in figure *I/O Copying*.

The memory area to which signals are copied is kept in Area objects. There is one class of Area objects for each signal type.

Area objects are created automatically at runtime in the system volume.

I/O Copying



Digital input and output signals are read from and set out to the hardware by the process, which executes with the base frequency. The AbsValue and RawValue of Co -signals are also handled at the base frequency. On the other hand, the hardware of analog signals is read with a frequency specified by the IOHandler object.

Tip If you have a heavily loaded process station, it is possible to relieve it by reducing the base frequency.

Graphical PLC Programming

This chapter describes how you create PLC programs.

Note! To get information on particular objects used in the PLC Editor, please refer to PROVIEW/R Objects Reference Manual.

The PLC programs are created with the graphical PLC Editor described on page ....

The Editor

You enter the plc editor from a PlcPgm object in the plant configuration. Select the object and activate *Functions/Open Program* in the menu. The first time the program is opened, you will find an empty document object. The program consist functions blocks and Grafcet sequences.

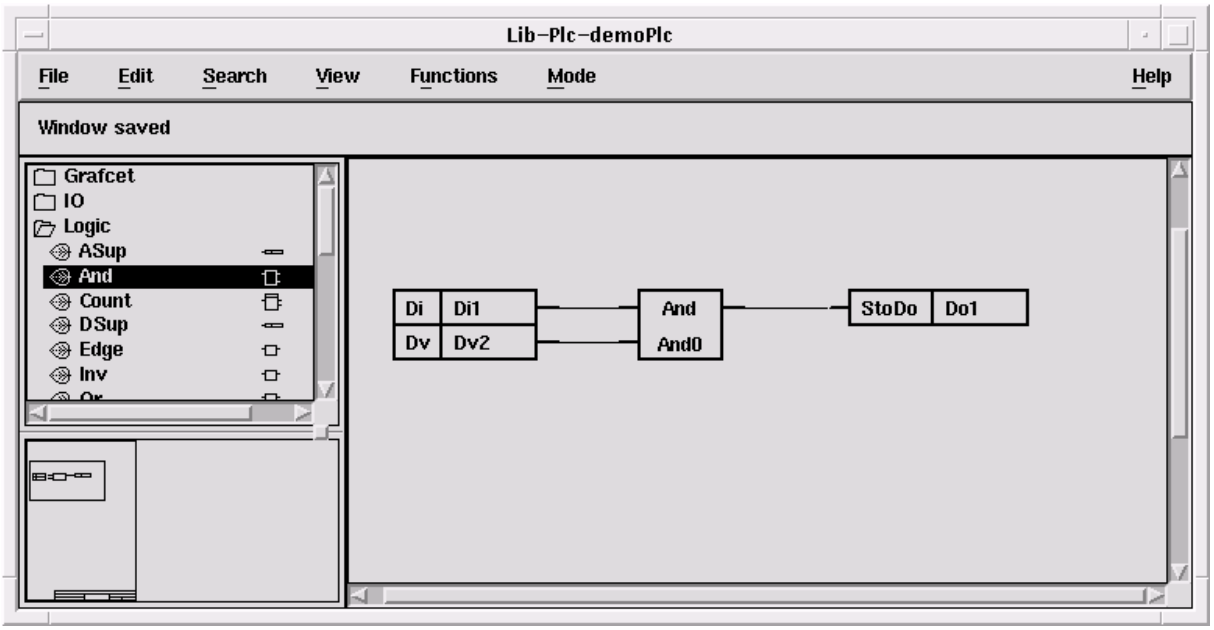
Programming with function block is made in a horizontal net of nodes and connections from left to right in the document. Signals or attributes are fetched on the left side of the net, and the values are transferred via connections from output pins to input pins of functions blocks. The function blocks operate on the values, and on the left side of the net, the values are stored in signals or attributes.

Grafcet sequences consist of a vertical net of nodes and connections. A state is transferred between the steps in the sequence via the connections. Grafcet and function block nets can interact with each other and be combined to one net.

Edit function objects

The plc editor consists of

- a working area.
- a palette with grafcet objects and function blocks.
- a navigation window, from which the work area can be scrolled and zoomed.



A function object is created by selecting a class in the palette, and pressing MB2 in the working area.

Modify the object

The objects is modified from the object editor. This is opened by selecting the object and activate *Functions/Open Objects* in the menu. Values of the object attributes is modified with *Functions/Change value* in the object editor menu. If an input or output is not used it can be removed with a checkbox. There is also a checkbox which states that the value of a digital input should be inverted.

Connect function objects

A output pin and a input pin is connected by

- Place the cursor on the pin, or in an area in the function object close to the pin, and press MB2.
- Drag the cursor to the other pin, or to an area in the function object close to the pin, and release MB2.

A connection is now created between the function objects.

Fetch a signal value

The value of a Di signal is fetched with a GetDi object. The GetDi object has to point at a Di signal and this is done by selecting the signal in the plant configuration, and then press Ctrl and double click on the GetDi object. The name of the signal is now displayed in the drawing. Dv signals, Do signals and attributes are fetched in the same way, with GetDv, GetDo and GetDp objects.

Store a value to a signal

The value of an output from a function object is stored in Do signal with a StoDo objects. The StoDo object is connected to a Do signal in the same way as the Get objects. Dv signals and attributes are stored with StoDv and StoDp objects.

Grafcet Basics

This section gives a brief introduction to Grafcet. For a more detailed description, please read a reference manual on Grafcet. Grafcet is an international norm or method to use at sequential control.

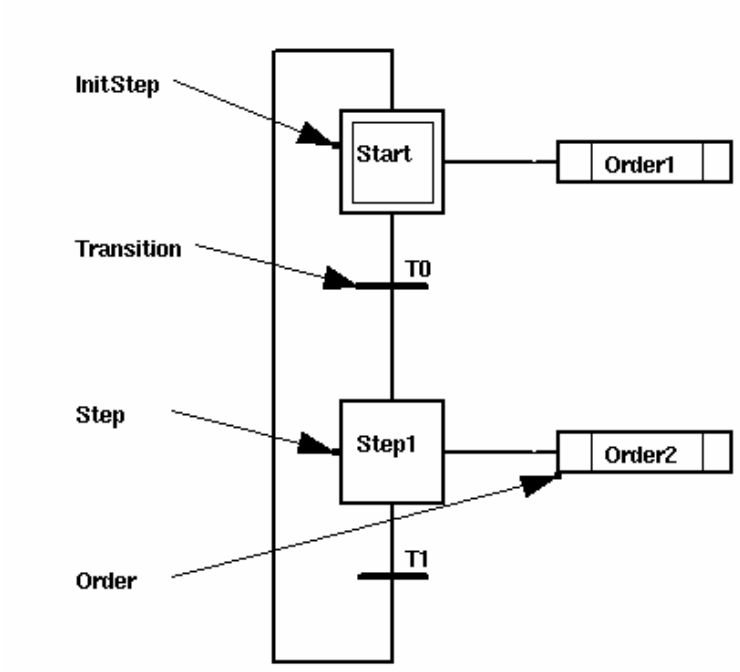
Grafcet consists of a number of steps, and to each step one or more orders are connected, which will be executed when the step is active. In order to move from one step to another, a transition is used. For each transition you have transition conditions, and the move can only take place when the transition conditions have been fulfilled.

Single Straight Sequence

We look at the single sequence below and assume that the step is active, which means that the order connected with the initial step will be carried out. This order will be carried out, until the initial step becomes inactive. Step 1 becomes active, when the transition condition for transition 1 has been fulfilled. Then the initial step becomes inactive.

A Grafcet program is always a closed sequence.

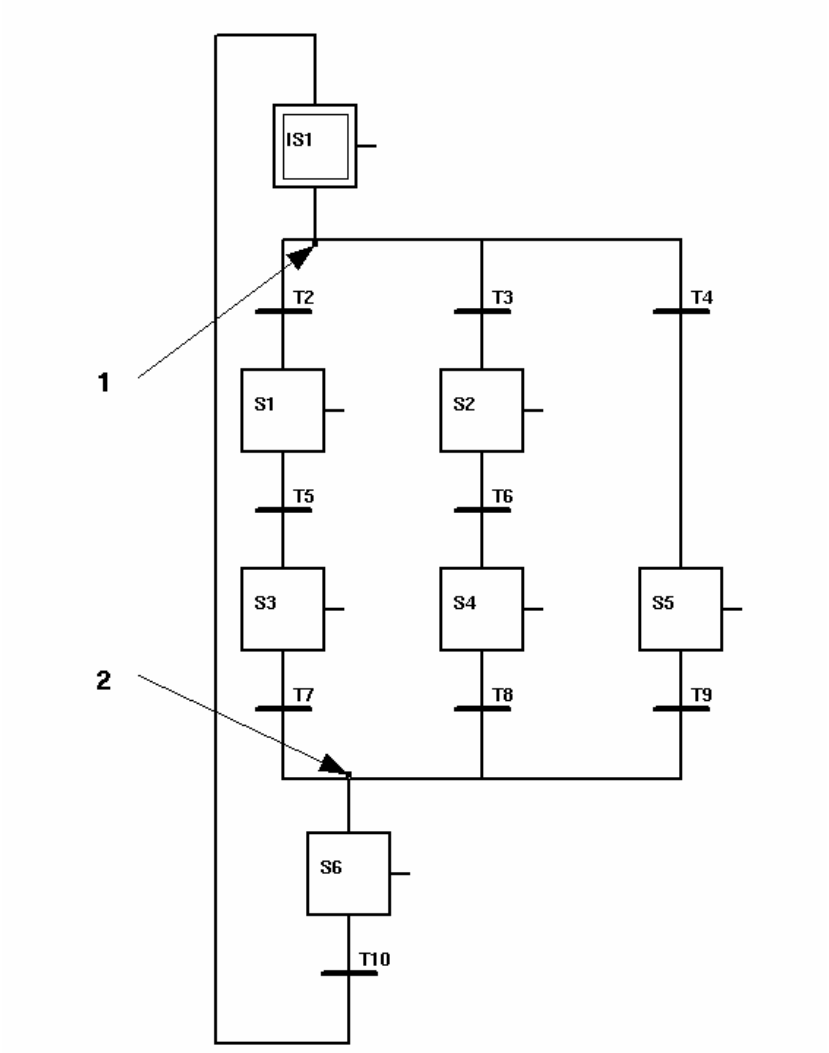
A Simple Straight Grafcet Sequence



Sequence Selection

A straight sequence is the most simple variant of sequences. Sometimes you may require alternative branches in your program, for instance when you have a machine, which can manufacture three different products. At the points where the production differs, you introduce alternative branches. An alternative branch is always followed by a connection of the different branches.

Sequence Selection



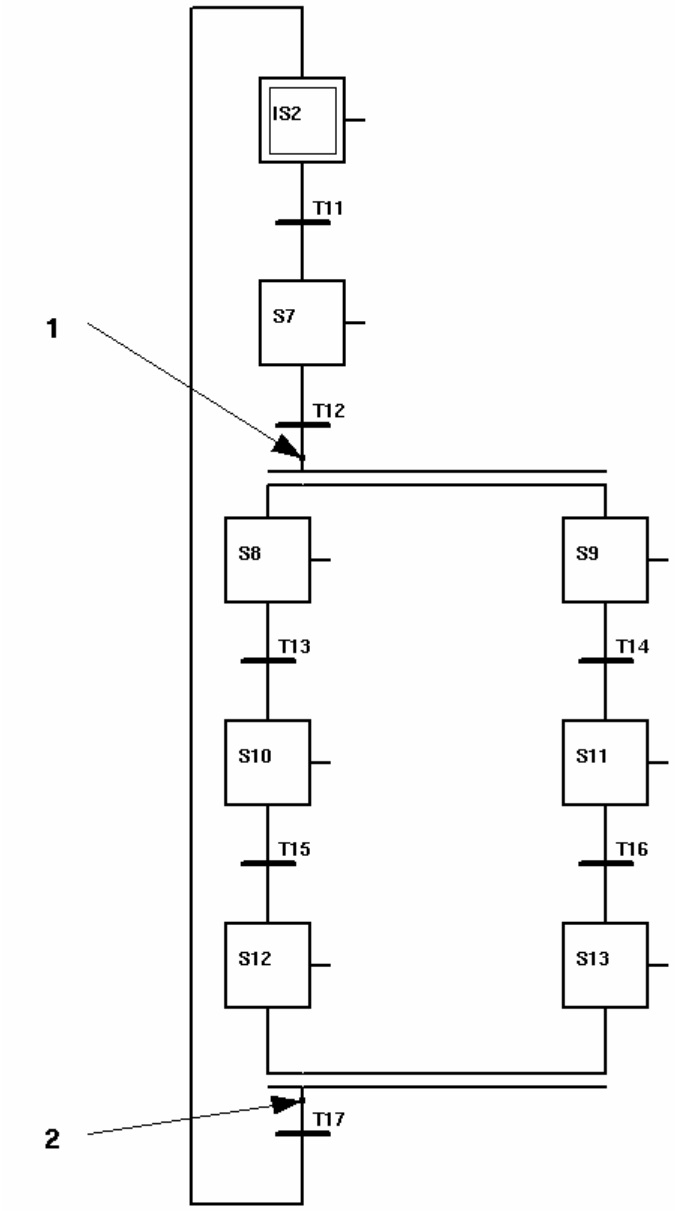
The example in figure See Sequence Selection shows the sequence for a machine which can manufacture the three products,e.g. Red, Green, and Blue. This is what happens at the different points:

At this point you select desired branch depending on which product you have chosen. When you have alternative branches, it is necessary always to write at the scantime the transition conditions, to make sure that only one can be fulfilled. If two transition conditions should be true, it is the chance that decides which branch to choose. However, only one branch will be executed.
Here the different branches are connected.

Parallel Sequences

Sometimes it may be necessary to start several parallel working procedures at the same time. It must be possible for these working procedures to operate independent of each other. In order to do this, parallel sequences are used.

Parallel Sequences



The example in figure See Parallel Sequences illustrates the sequence for two machines, which are drilling two holes at the same time and independent of each other. The following happens at the different points:

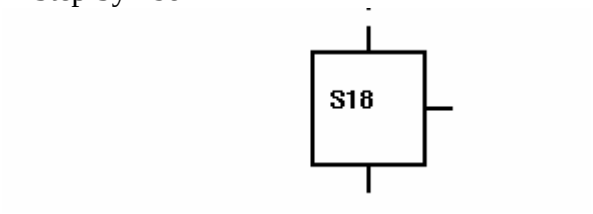
- When the locking is ready, the two machines start drilling. They carry out their drilling independent of each other.
- When both machines have finished drilling, the parallel branches are connected to a straight sequence.

Step

A Step is used to describe a state in the process. The following applies to a step:

- A step can be active or inactive.
- An attribute, Order , indicates whether the step is active or not.
- You can connect one or more orders to a step.
- The step can be allocated a name at your own discretion.

A Step Symbol

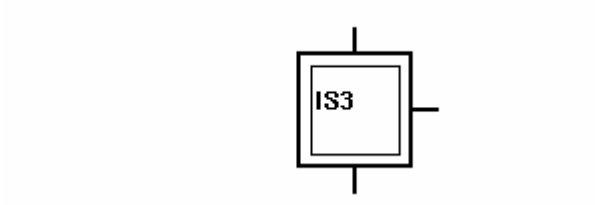


Initial Step - InitStep

In each sequence you must have an initial step (InitStep) which differs from a usual step at the following points:

- Only one initial step is permitted in a sequence.
- When the program starts its execution, the initial step is active.
- An attribute, Order , indicates whether the step is active or not.
- You can always make the initial step active by setting the reset signal of theinitial step.

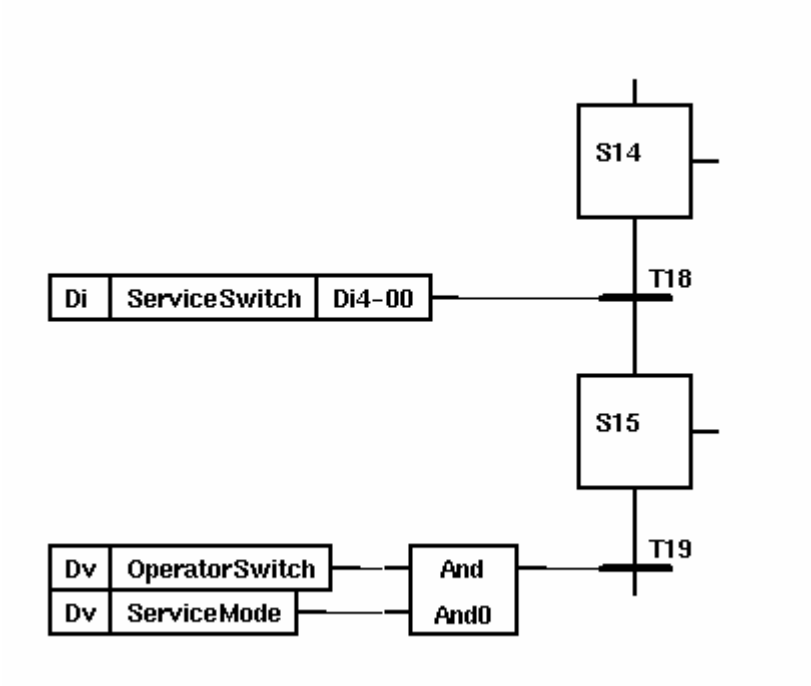
An InitStep Symbol



Transition - Trans

As mentioned above, the transition (Trans) is used to start a transition between an active and an inactive step. A logical condition, for instance a digital signal, is connected to a transition.

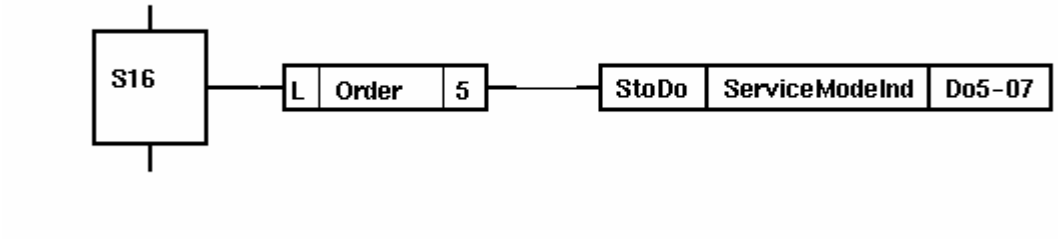
A Transition Example



Order

It is possible to connect one or more orders (Order) to each step.

An Order Example

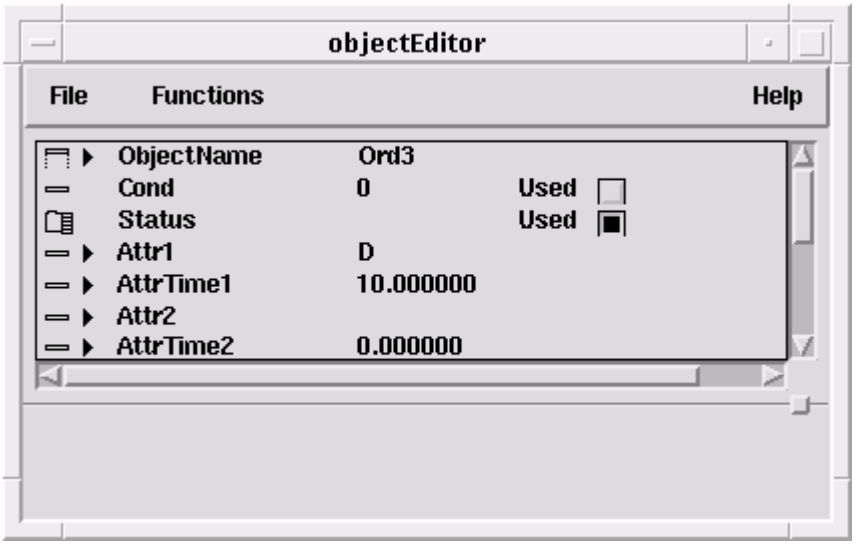


Normally the output is active when the input is active, but for each order you have a number of attributes, with which you can influence the function of the output:

- D Delay
- L Time limit
- P Pulse
- C Conditional
- S Stored

These functions are described in detail in PROVIEW/R Objects Reference Manual . The principles are that you indicate the name of the attribute (capital letters) and possible time by means of the Attribute Editor. The figure below illustrates how to delay an order from being active for 10 seconds.

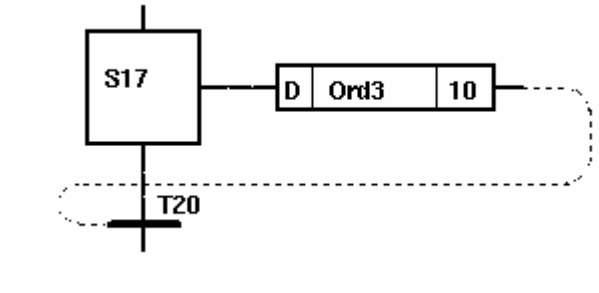
Order Attributes



The selected order attributes are written in the order symbol.

The figure below illustrates how you can use an order object with delay to make a step active for a certain time.

A Delayed Transition

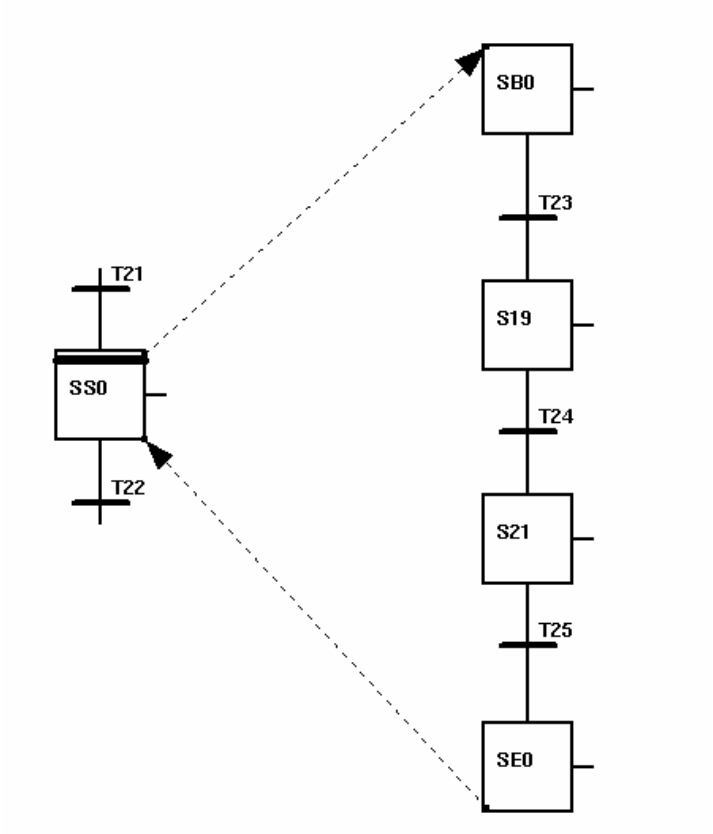


Note! You must use a ConFeedbackDigital connection to connect the delayed order object with the transition object, otherwise the order of execution will not be correct. See Feedback Connection on page See Feedback Connection for further information.

Subsequence

When creating complex Grafcet programs, it is often an advantage to use subwindows, and in these you place subsequences. In this way you get a better arranged program

Subsequence



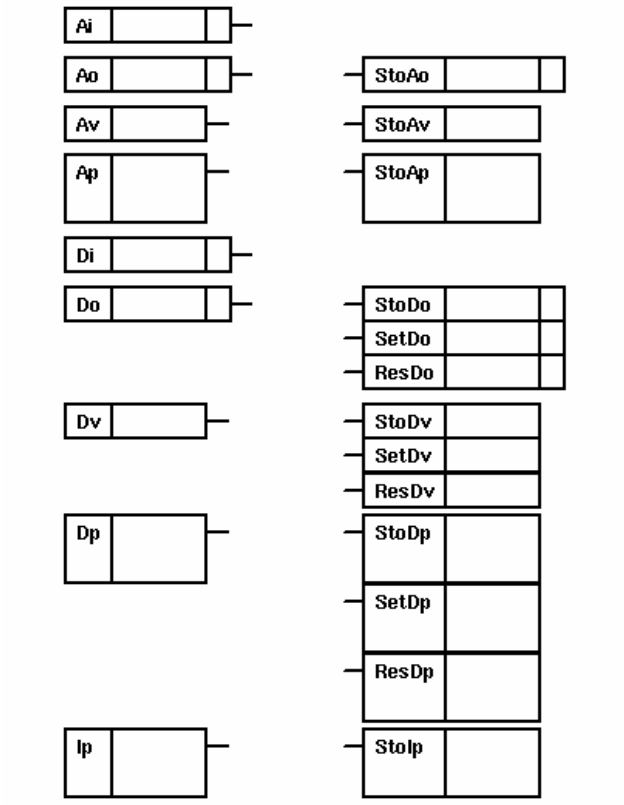
The above figure shows the sub sequence of a SubStep . A sub sequence always starts with an SsBegin object, and ends with an SsEnd object. In its turn a subsequence can contain subsequences.

An Introduction to Function Block Programming

Input and Output blocks

Input and output blocks are used to read and write values. You have input and output blocks for each signal type. The figure below shows a number of input and output blocks. These are placed in the folder IO.

Input and output blocks



To read signals you use GetAi , GetDv or GetAo . When you want to give a value to a signal, you use for instance StoAv , StoDo , SetDv or ResDo .

Digital values can be written in two ways:

- Sto stores the input value, i.e. if the input is 1 the signal becomes 1, and if the input is zero the signal becomes zero.
- Set sets the signal to one if the input is true, Res sets the signal to zero if the input is true. For instance, if you set a digital output signal with a SetDo object, this will remain set until you reset it with a ResDo object.

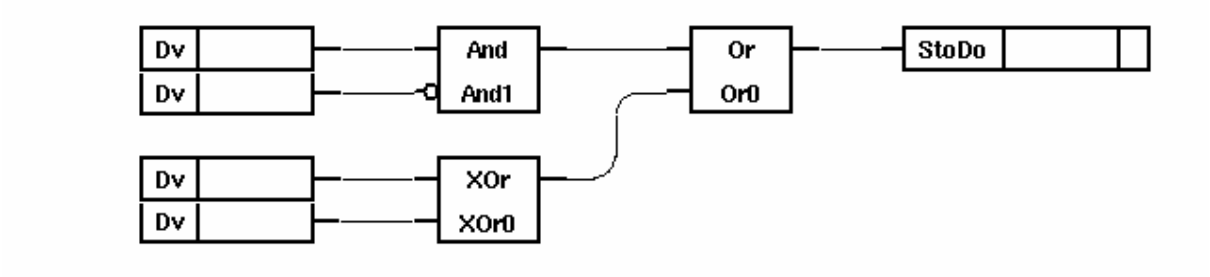
To read, respectively assign attribute values other than the ActualValue attribute, you use the following:

- analog attributes, GetAp and StoAp
- digital attributes, GetDp and StoDp

Logic Blocks

A number of objects are available for logical programming, for instance And- gate (And), Or-gate (Or), inverter or timer. For the logical programming digital signals are used. The objects are placed in the folder Logic.

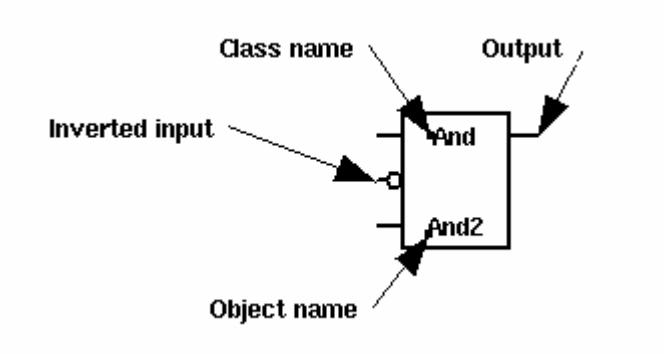
Logics Blocks



The figure below shows an And-gate. For this object the following applies:

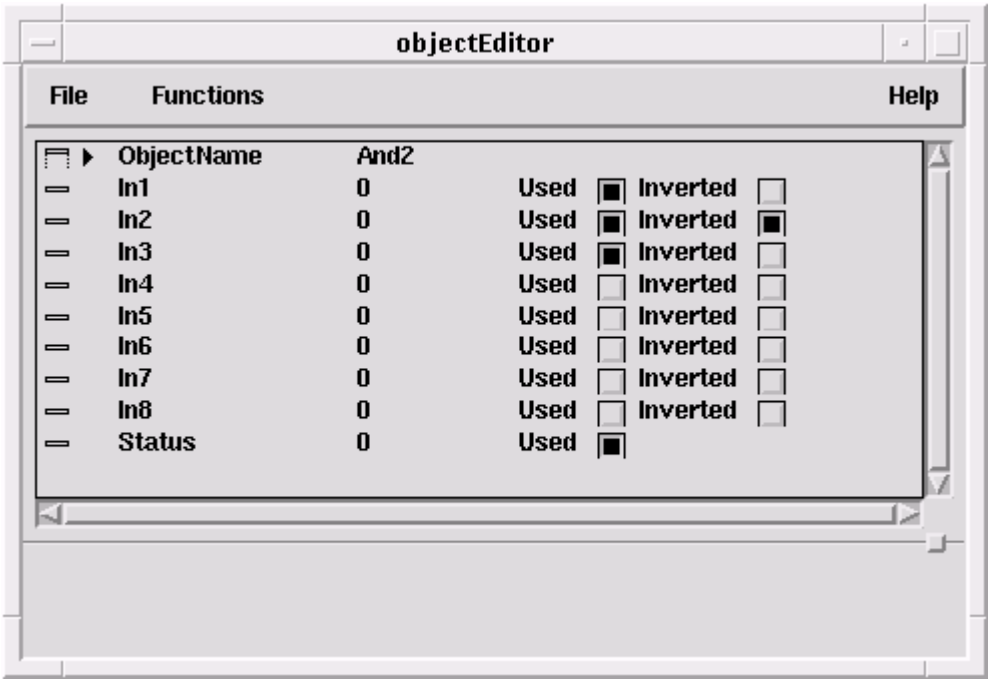
- Inputs to the left
- Output to the right
- Class name is written at the top
- The object name is written at the very bottom (can be changed by the user)
- You can use a variable number of inputs, default is 2.
- The inputs can be inverted, indicated by a ring on the symbol's input.

And-gate



The attributes of the And-gate are changed with the Attribute Editor.

Attributes of the And-Gate

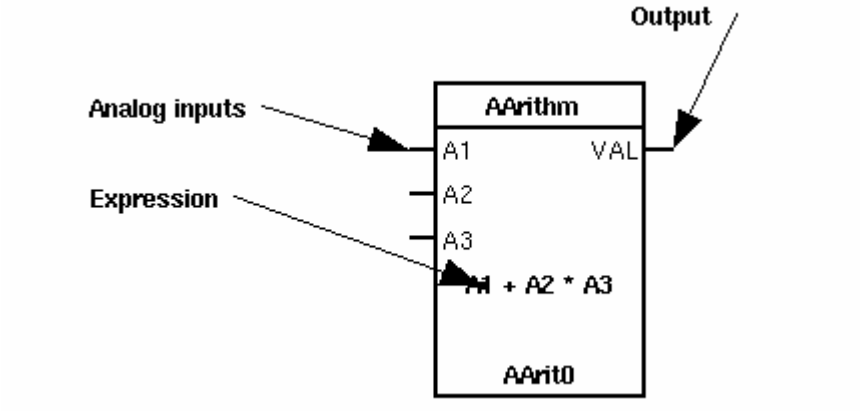


The other objects in the Logic folder have similar parameters, see PROVIEW/R Objects Reference Manual .

Calculation Blocks

The folder Analog contains a number of objects for handling analog signals, for instance filters, summation blocks, and integrators.

Arithmetical Calculation Block

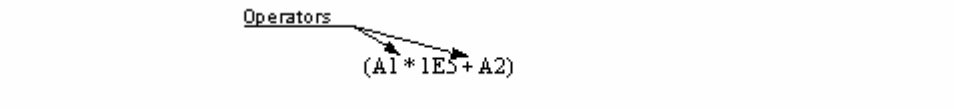


In this guide we do not describe the function of the objects, but it may be expedient to comment on the use of arithmetic blocks. The blocks are used for calculation of user defined expressions. These are written in the C language.

In figure See Arithmetical Calculation Block the block will calculate the expression (A1 + A2 * A3) and give the output this value. A1, A2 and A3 are representing analog values, for instance signals supposed to be connected to the inputs of the object.

When writing these expressions it is important to use space before and after the operators, otherwise the expression may be misinterpreted at the execution.

An Expression



Alarm Supervision

In PROVIEW/R it is possible to supervise analog and digital signals against a limit value. If the supervised limit is exceeded, an alarm is sent to the Message Handler, who in his turn sends the alarm to the out unit, e.g. an operator dialog.

See PROVIEW/R Objects Reference Manual regarding the attributes of the objects.

Supervision of Digital Signals

For supervision of a digital signal or attribute, you use the DSup object (Digital supervisory), which is in the folder Logic.

When a signal or attribute will be supervised, you connect the object to the desired signal by a connection line.

Digital Supervisory Objects

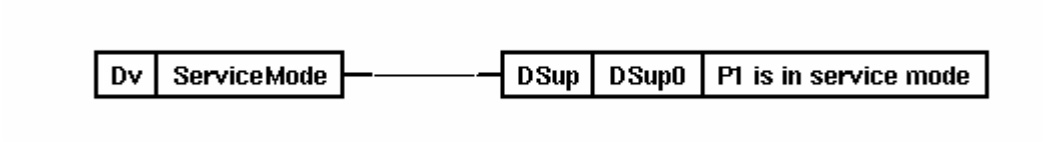


Figure Digital Supervisory Objects illustrates supervision of a Dv signal and a digital attribute.

You also have an attribute in the DSup object, CtrlPosition , that indicates whether the alarm will be activated when the supervised signal/attribute becomes true or false.

Supervision of Analog Signals

For supervision of an analog signal or parameter, you use the ASup object (Analog supervisory), which is in the folder Logic.

Supervision takes place in the same way as for DSup objects with the exception that you can choose whether the alarm will be released when the supervision limit is not reached or exceeded.

Compile the plcpgm

Before starting to compile, you have to state on which platform (or platforms) the volume of the plc should run. Open the volume attributes editor from the navigator menu: File/Volume Attributes, and enter the OperatingSystem. Note, that more than one operating system can be choosen. The volume can at the same time be run in the production system, in a simulation system and in a educational system, and the systems can have different platforms.

Now, the plcgpm is compiled, by choosing File/Compile in the plc editor. Any waring or error messages will be displayed in the message window.

Creating Process Graphics

This chapter describes how you create process graphics.

Process graphics are drawn and configured in the Ge editor.

The Ge editor

Ge is opened from the menu in the navigator: *Functions/Open Ge.* It consist of

- a tool panel
- a work area
- a subgraph palette
- a color palette
- a window displaying the plant hierarchy
- a navigation window

Background picture

A background image is drawn with base objects such as rectangles, circles, lines, polylines and text. These are found in the tool panel. Create a base object by activating the pushbutton in the tool panel and dragging or clicking MB1 in the work area. If the base object should be filled, select the object and activate fill in the tool panel. Change the fillcolor by selecting the object and click on the desired color in the color palette. Change the border color by clicking with MB2 in the color palette.

Subgraphs

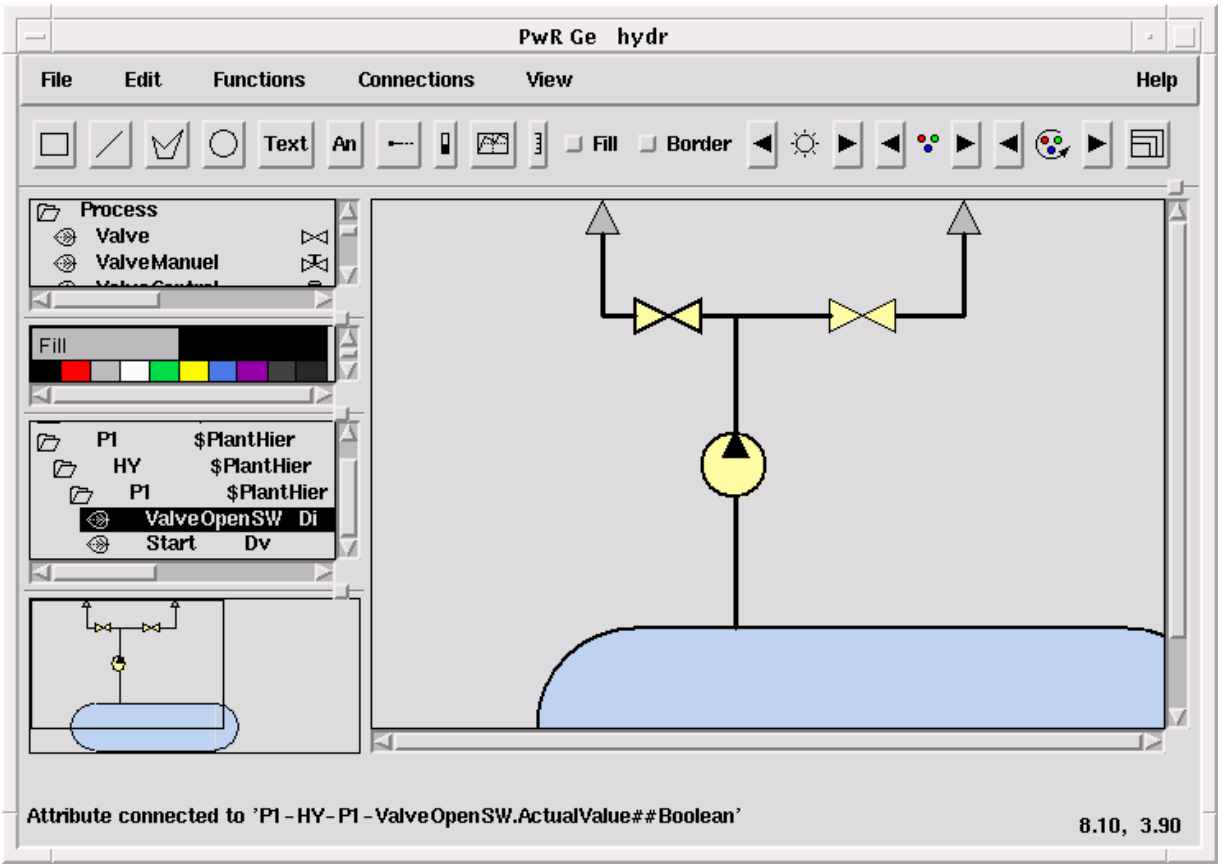
A subgraph is a graphic component, e.g. a valve, a motor, a pushbutton. To create a subgraph, select a subgraph in the subgraph palette and click MB2 in the work area.

Groups

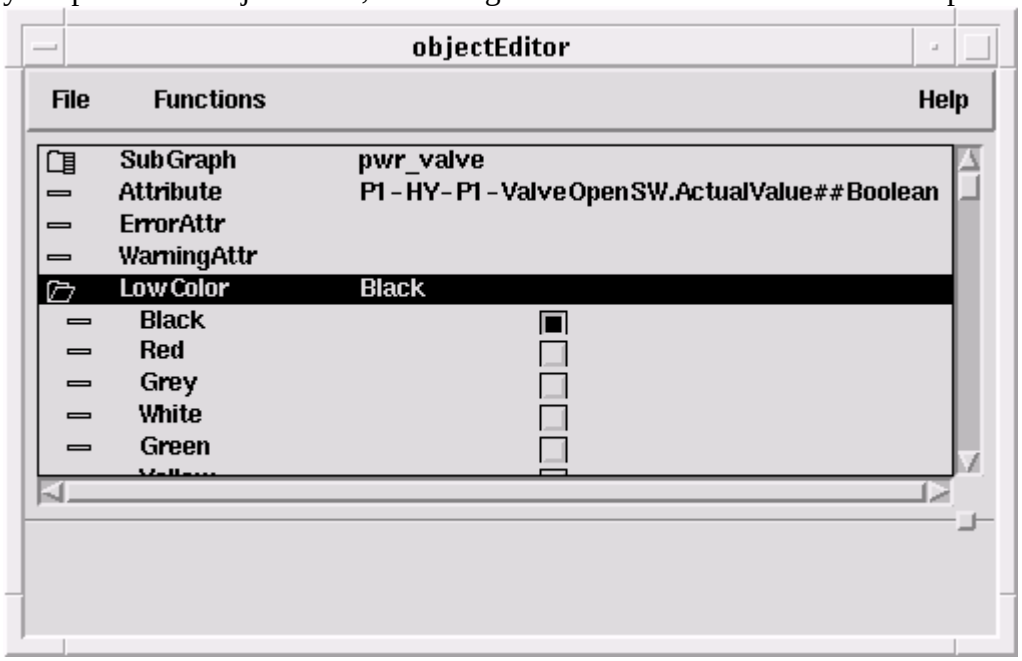
Base objects and subgraphs can be grouped together by selecting them and activating *Functions/Group* in the menu.

Dynamics

Subgrpahs and groups has dynamic properties, i.e. they can be connected to signals in the runtime database, and change color, position or shape depending on the values of the signals. A subgraph often has default dynamic behaviour, for example a valve shifts between two colors. You only have to connect the valve to a digital signal to make it work. This is done by selecting a signal in the plant hierarchy window, and click on the valve with DoubleClick MB1.

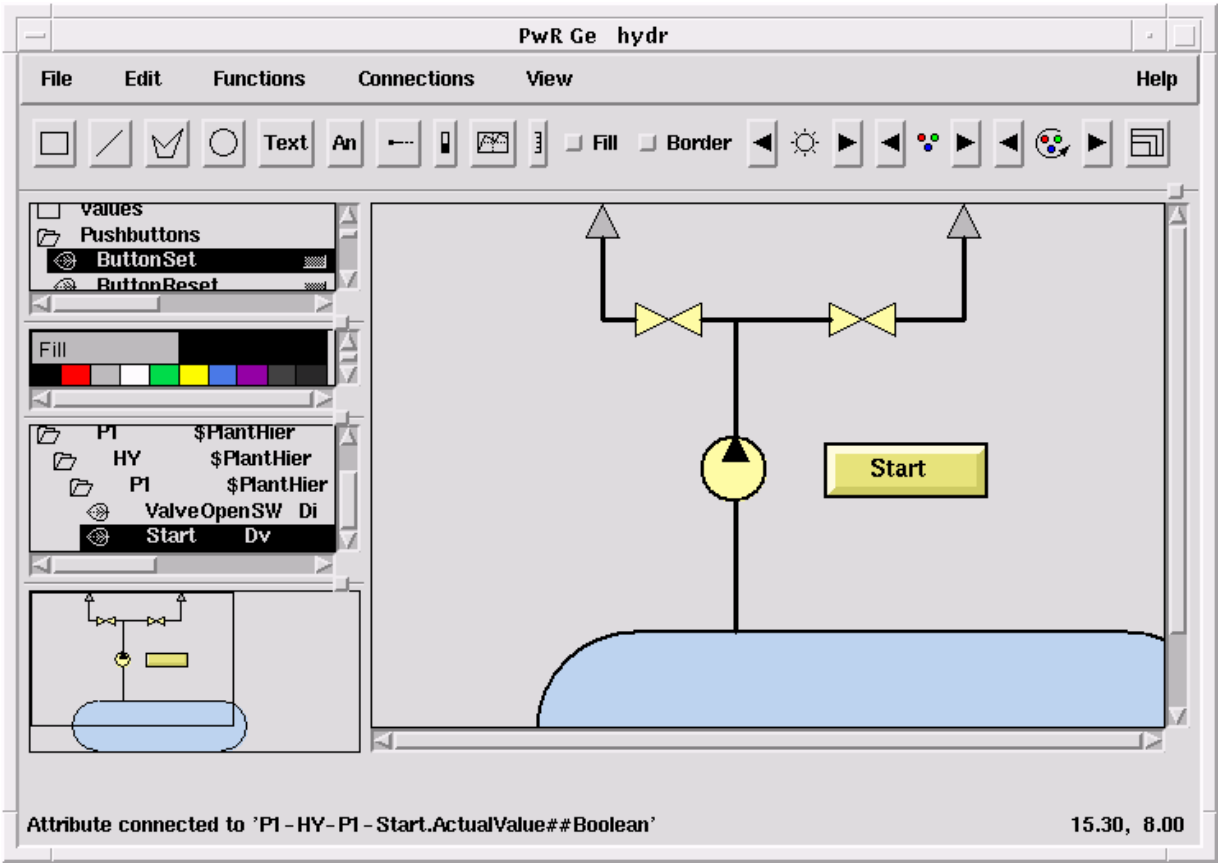


The default color the valve will gain when the signal is low, is white. If you want to change this, you open the Ge object editor, and change LowColor from Inherit to for example black.



The high color. i.e. the color the valve has when the signal is high, is the color that the valve is drawn with in the editor. This is changed in the ordinary way, by selecting the object and click in a color in the color palette.

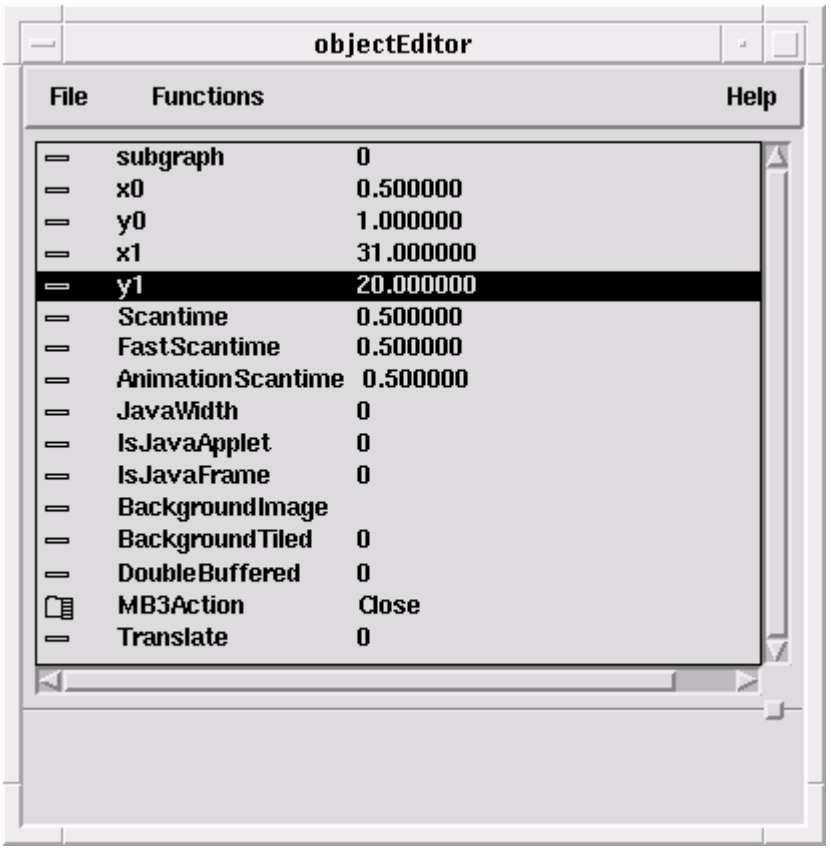
A pushbutton has an action property, is sets, resets or toggles a signal in the database. A button with a set action is created by selecting a ButtonSet in the subgraph palette and click MB2 in the work area. The signal that should be set is connected as above, by selecting the signal and click with Shift DoubleClick on the button. In the object editor, a button text can be assigned.



Group also has a dynamic property, i.e. they can shift color, move, or perform some action. They don't have any default action or default color, as the subgraphs. You have to assign this for each group.

Graph borders

The drawing area in Ge is unlimited in every direction, so before saving the graph, you have to state the borders of the graph. Open the graph attributes with *File/Graph attributes* in the menu. Measure the coordinates of the upper right corner, and insert as x0 and y0, then measure the coordinates of the lower left corner and insert as x1 and y1. The measurement is done by placing the cursor in position and read the coordinates in the ge message row.



Finally save the graph with *File/Save* in the menu. A file of type .pwg is now created in the $pwrp_pop directory.

Configuration in the workbench

The XttGraph object

To each plant graphics belongs a XttGraph object. This object is usually a child of the operator place object (OpPlace) for the node, on which the graphics will be displayed. It is necessary to create a graph object for each node, on which the graphics will be displayed. However, you only need to have one graph file. The following attributes in the graph object must be set to appropriate values:

- Action, the name of the pwg file with file type, e.g 'hydr.pwg'.
- Title, the title of the graph window.
- ButtonText, text of the button in the operator window.

The XttGraph object contains other attributes which e.g. helps you to customize the position and size of plant graphics. These attributes are described in detail in PROVIEW/R Objects Reference Manual .

Running and Testing a Proview System

In preceding chapters we have described how to configure a PROVIEW/R system, how to create PLC programs and how to create plant graphics. Now it is time to run and test the system.

This chapter shows how to:

- create load files
- create a boot file

Syntax control

Before creating load files it is appropriate to make a syntax control. This is done from the menu in the navigator: *File/Syntax*.

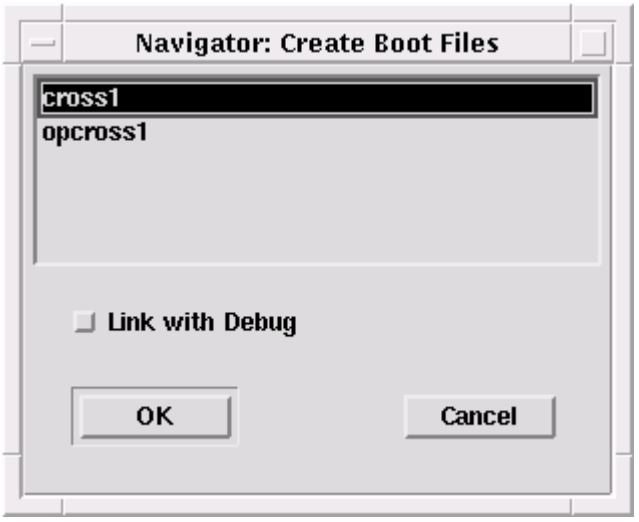
Creating Load Files

Before starting the runtime environment you have to create a number of load files, one for each volume in your system. You create the load files from the Navigator of each volume:

Choose *Functions/Create Load Files...*

Creating Boot Files

For every node in the project you have to create a boot file. The bootfile contains mainly the root volume to load for the node.



At this point, everything in the development environment is configured and generated, and its time to set up the runtime environment.

QCOM Bus