ProviewR

OPEN SOURCE PROCESS CONTROL

Release Notes V6.1

2023 01 20

Table of Contents

Upgrading to ProviewR V6.1.0

This document describes new functions in ProviewR V6.1.0, and how to upgrade a project from V6.0.0 to V6.1.0.

Updates

PROFINET

The PROFINET configuration tool has gone through a major overhaul. As has the runtime implementation.

PROFINET Configurator

The PROFINET configuration tool has been updated to better handle more recent versions of the GSDML specification. A few noteworthy changes:

- Updated GSDML conformance level.
- Configurator will never delete channels for you unless the Module Class changes. Instead it will notify the user that it couldn't populate a certain module/device and leave it to the user to remove them if the user want them generated by the configurator.
- The DAP is now populated as any other module is, as it's possible for a DAP to carry data.
- The generation of module and channel objects have been updated to name the signals in accordance to the GSDML data where possible using a best effort approach since we are still limited to how many characters we can use, and the GSDML is not.
- Runtime XML files have been updated to a slightly different schema. The upgrade procedure includes a step to convert these files. Most notable is that the tool now uses the ID to map modules/submodules to their respective slot/subslot instead of an index as was the case previously. The result is that it's now much more safe to switch between GSDML files for a device since the ID should (according to specifications) never change but the index order in which they appear can.
- It loads faster.
- Old runtime xml files are now deleted when the corresponding object is deleted, hopefully leaving you with a less bloated $pwrp_load folder.
- It now uses category menus for modules where implemented in the GSDML. Unfortunately, most vendors don't implement this neat feature.
- Input validation. It shouldn't be possible to input erroneous formatted data in the input fields anymore as all input is validated against either GSDML or other trivial specs like the format of an IP address or device name. In cases where the GSDML does not state anything the datatype is the limiting factor. Also, all input fields or selectable items now show some help about the item/field in question, hopefully making it easier for the user.

- SendClock/Reduction ratio is now populated according to GSDML and only falls back to defaults if not present in the GSDML. Previously only specification defaults where shown making it possible to choose a combination not supported by the device.
- You can now choose to skip the IP assignment leaving this to another controller or supervisor.
- Copying of slots should now be more robust and not crash on you in some situations.
- More diagnostics are saved in order to display accurate error messages in runtime.
- Phases are now implemented and can help reduce congestion during, for instance startup in large installations.

And a lot more minor fixes that should make it feel more mature.

PROFINET Runtime

The runtime part of the PROFINET implementation has been updated. Alongside these changes the default graph for the PnDevice class was also updated.

The old dat files describing different vendors have been replaced by one single profinet_devices.xml file which can be generated by the binary pn_get_deviceid should the need arise, do note that one would need access to the Profibus Foundation for their list of manufacturers. The Profinet Viewer use this file to display more helpful information about found devices.

Asynchronous Read/Write

Asynchronous read/write are now in place. Previously you could do a write but without any means of getting the result back. Both Read and write are now implemented. They also have their own subgraph within the PnDevice graph as shown in Figure 1 and Figure 2. Each request can be programatically initiated by means of the SendReq attribute. The Status attribute (Ready/Busy) indicates wether we are ready for a transaction or not.

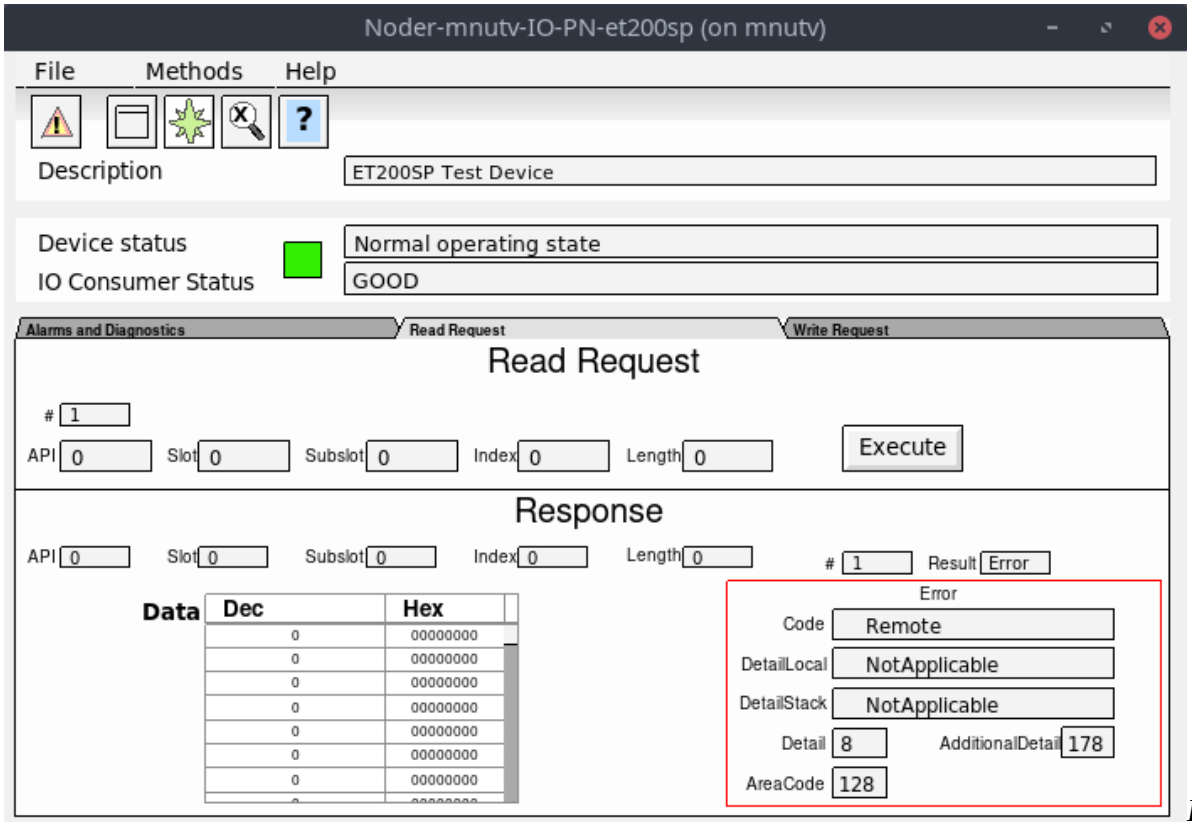


Figure 1: PnDevice Read Request after an erroneous read request

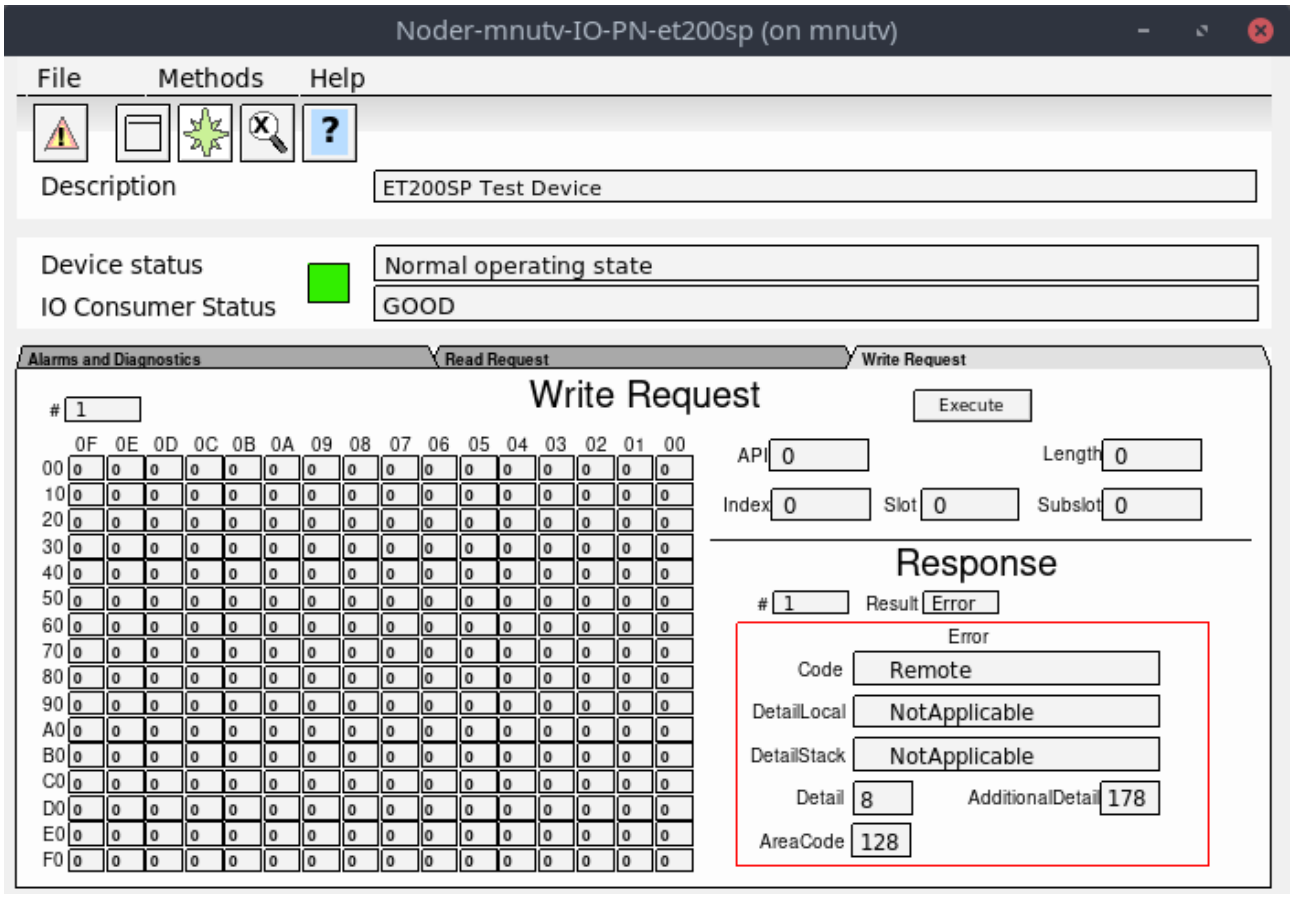
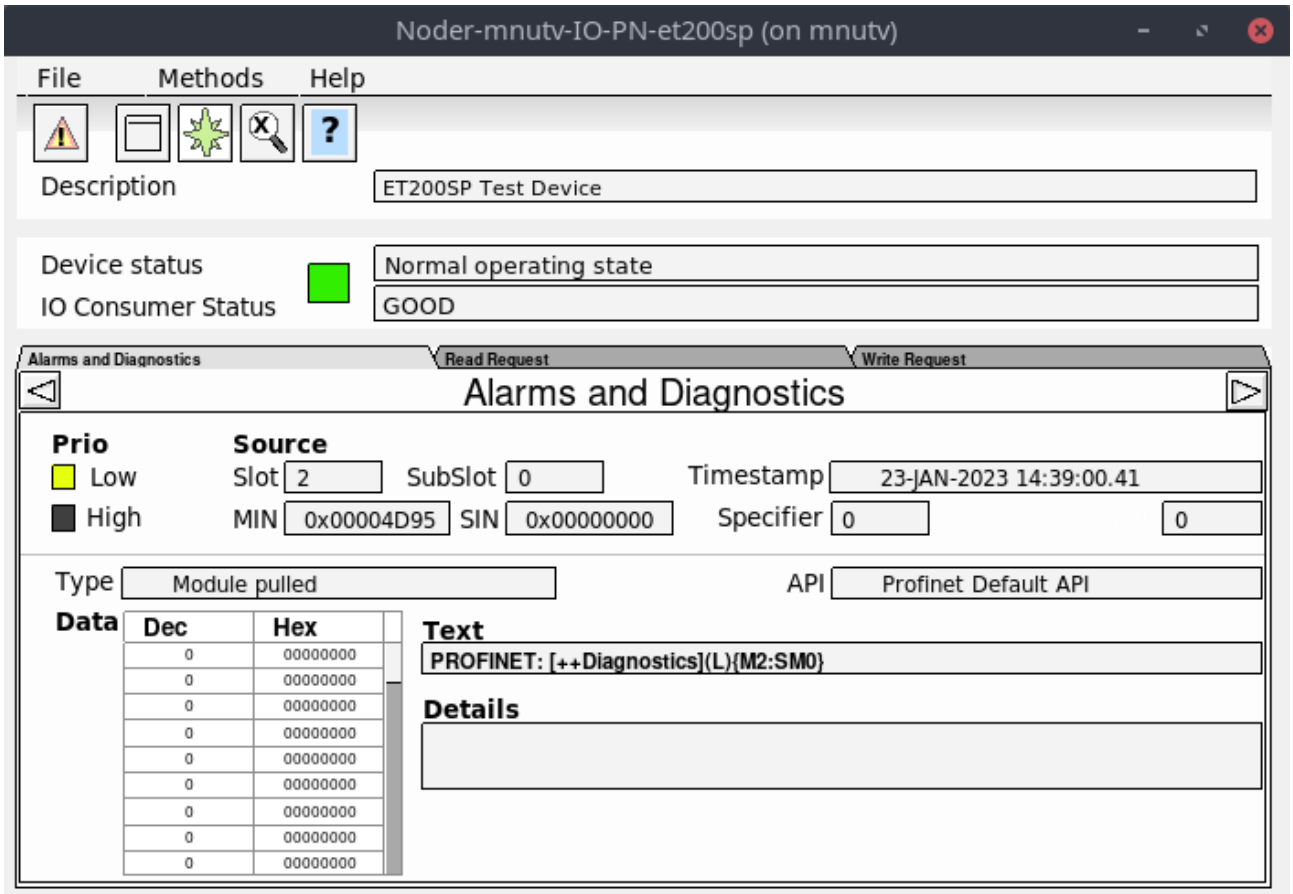


Figure 2: PnDevice Write Request after an erroneous write request

Alarms and Diagnostics

Each PnDevice can now be configured to generate appropriate alarms depending on severity. You can also select to log these to the ProviewR log file, or both! Within each PnDevice there's a mask (AlarmActionSelect) that selects what action to take. And each module inherits (by default) that setting which, if needed, can be overridden on module level.



$pwrp_cnf in runtime

This is more of a general update. The $pwrp_cnf environmental variable is now present in the runtime. This is now the default place where ProviewR runtime will look for GSDML files when no absolute path is given. Previously one had to place them in a folder which existed both in the development environment and in the runtime environment or in $pwrp_exe. Most often the files ended up in $pwrp_db, but having configuration files in $pwrp_db wasn't always the most elegant solution.

Ge

New dynamic properties

DigTransparency.SmoothTransition

DigTransparency.SmootTransition will cause a gradual transition of transparency over a number of scans.

Invisible.dim_level

The previous action for Invisible.dimmed has been to draw black lines or texts with gray color. By setting Invisible.dim_level to a value larger than zero, transparency will be used instead, and the dim_level value specifies the transparency level.

Rounded rectangle, thin shadow

Transparency has been disabled for rounded rectangles with shadow. If the new property thin_shadow is set, transparency is enabled.

Undo, redo

The journal is updated and handles several actions that previously was ignored.

Layout manager and script modules

Script modules is a way of dividing the area of a graph into sections that will be positioned by a layout manager dependent of the dimension of the current window. Preferred size, nearest neighbors and a priority is set on each module. For windows with reduced size, modules with low priority will be dismissed. The graph has to be written as a script and call the Layout() function.

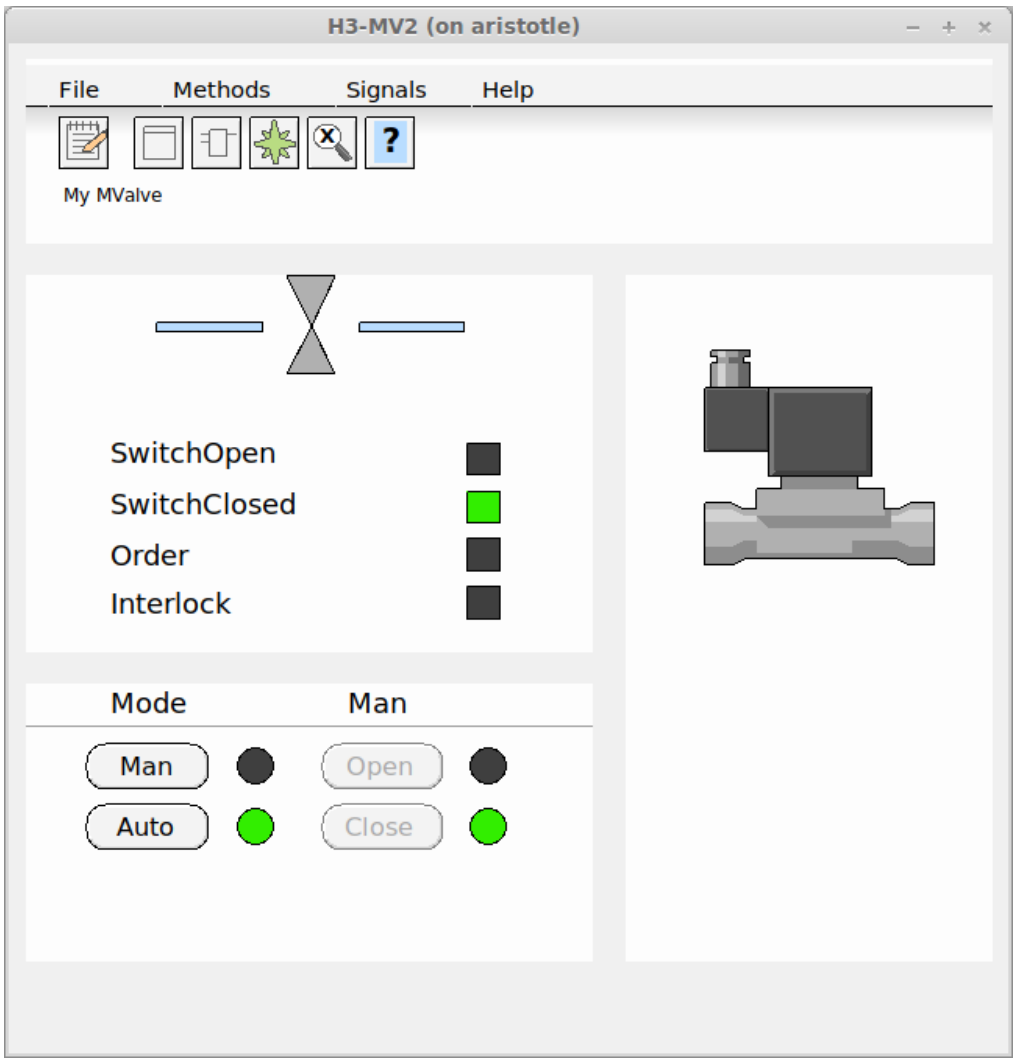


Fig Object graph, normal size

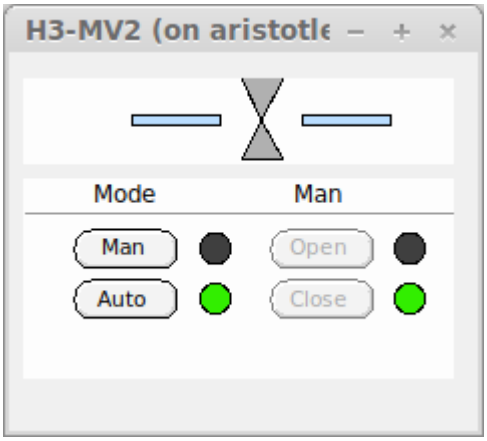


Fig Object graph, small size

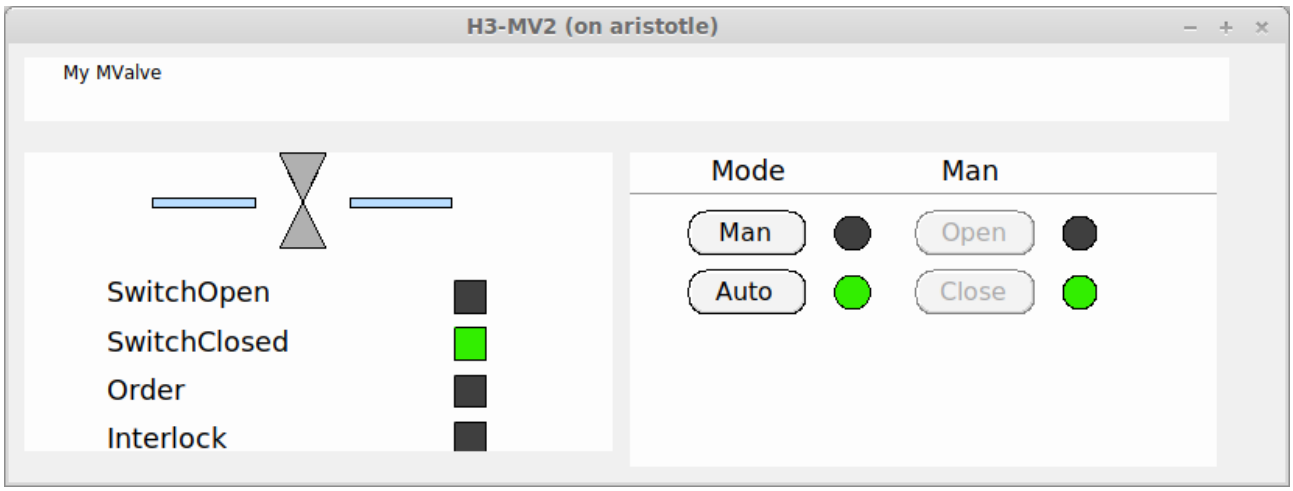


Fig Object graph, landscape format

Layers

A layer is plane or level in the editor where you can create, edit and organize graphical objects. Layers are stacked on top of each others, and controls the visibility and and the possibility to edit objects.

When a layer is set active, objects in the layer can be created and edited. Only one layer can be active at a time, and there is no risk to accidently move or edit objects in other layers. The activity is controlled from the lower window in the object tree view, the selected layer is set active. From this window, also the visibility of the layer is controlled. By clicking on the eye icon, the layer can be set visible or invisible.

In the upper window of the tree view, the organization of layers can be seen. Layer objects are position on the top level, and object in the layer are positioned under the layer object. By opening the layer object, dynamics for the whole layer can be set, eg transparency or color dynamics. The priority for color and transparency dynamics differs, and for colors, the dynamics for the individual objects have higher priority, wile for transparency, the dynamics for the layer has higher priority.

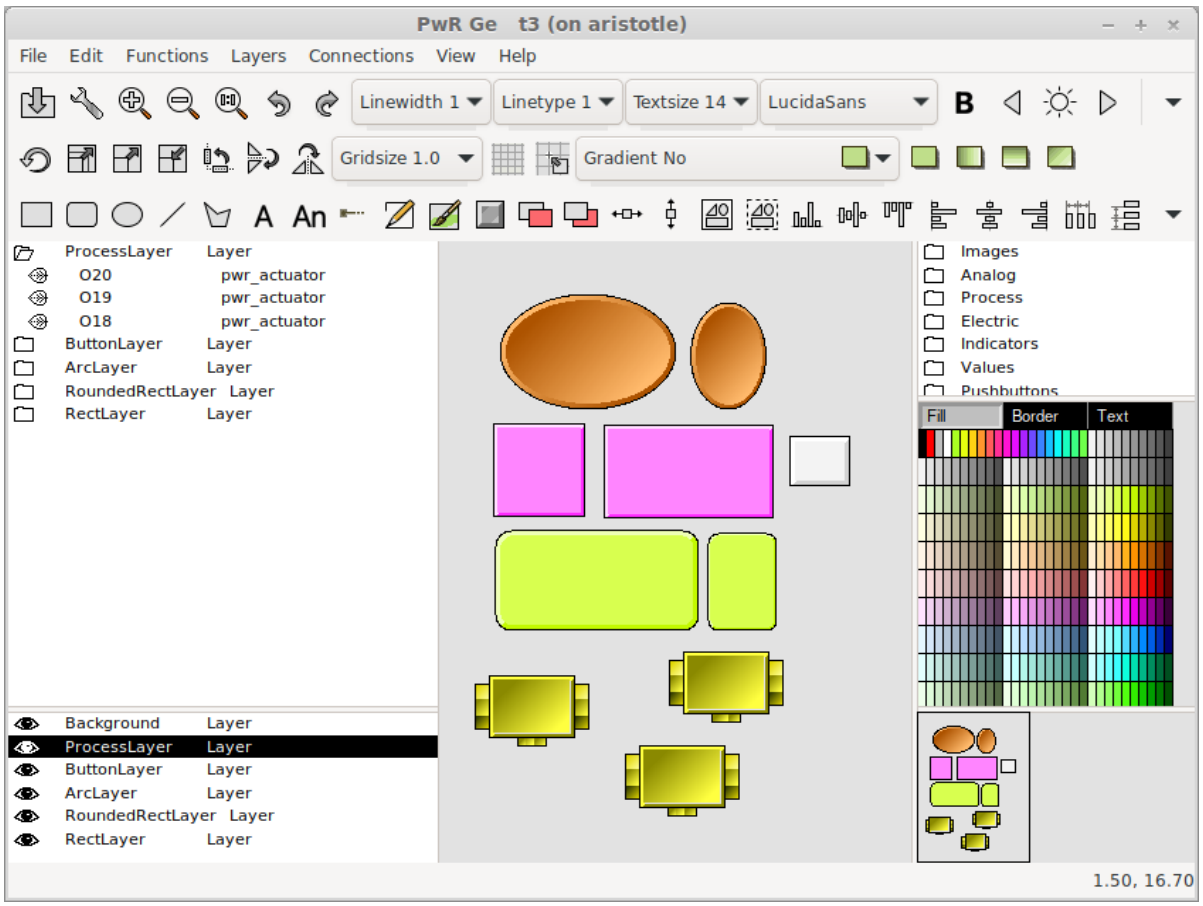


Fig Layers

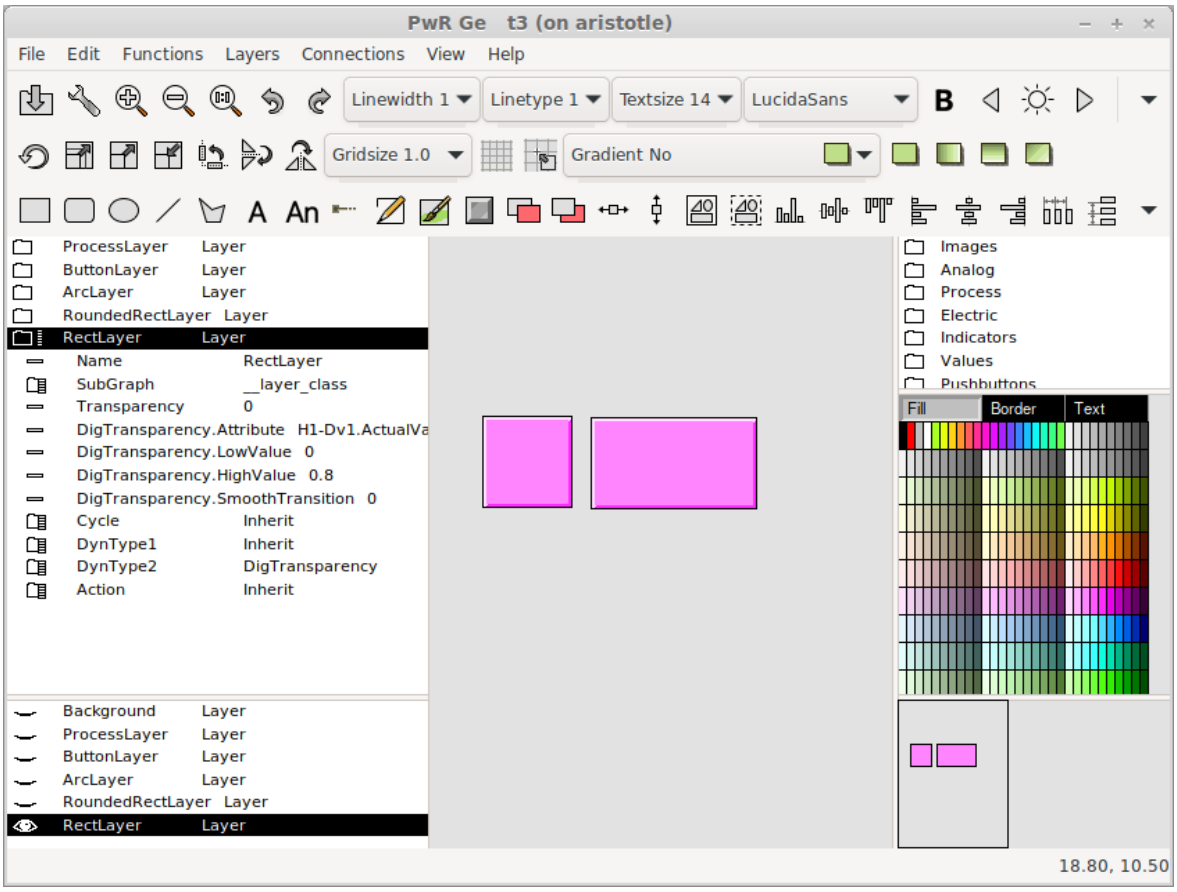


Fig Layer dynamic and visibility

From the Layers menu layers is created or deleted. It also contains functionality to move objects between layers and to merge layers can be found there.

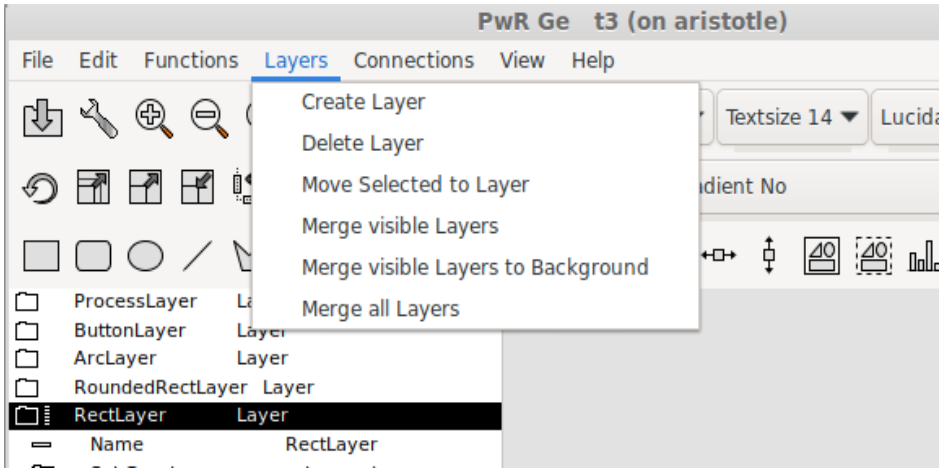


Fig Layers menu

Object graph defined with script

If an object graph is defined with a Ge script, this has to be specified in the PopEditor attribute in the $ClassDef object for the class. PopEditor should be set to *GeScript*.

New Ge script functions

SetObjectVisibility()

Set object visibility to visible, invisible or dimmed.

CreateLayer()

Create a new layer.

GetWindowDimension()

Get the current dimension of the window containing the graph.

Layout()

Calculate a layout for the graph depending on the window dimension and the priorities and sizes of the script modules.

SetGraphOptions()

Set graph options.

TranslateObjectName()

Translate an object name containing symbols (eg $object) to the real object name.

LayerSetActive()

Set a layer active.

LayerResetActiveAll()

Set all layers inactive.

MergeVisibleLayers()

Merge visible layers to one layer.

MergeAllLayers()

Merge all layers to background layer.

MoveSelectToLayer()

Move selected objects to the active layer.

LayerGetFirstObject()

Get the first object in a layer.

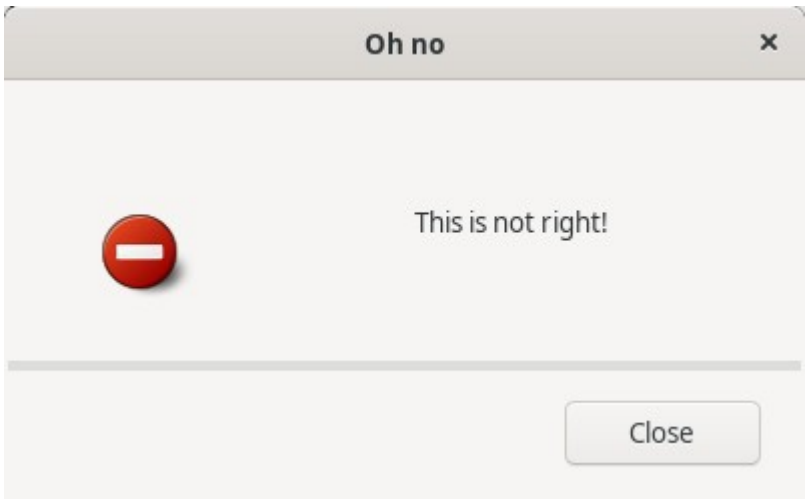
LayerGetNextObject()

Get the next object in a layer.

New Xtt script function

TextDialog()

Display a dialog with a text and an icon.



Configurator

Command 'show objid/hex'

/hexadecimal will show the object index in hexadecimal form which is convenient when examining flow files and object modules where the object index is part of the file name.

Class volume editor start

The class volume editor can be started from the prompt with 'pwrs -c'. Now the fil extension and directory can be omitted, eg

> pwrs -c cvolxxx

Class volume additional hierarchy

One additional hierarchy level can be added between the Class object and ClassDef objects in class volumes. It's a suitable way to for example group classes belonging to a component or aggregate, or group classes describing a complex class with several internal classes.

The additional hierarchy object should be of class $ClassHier.

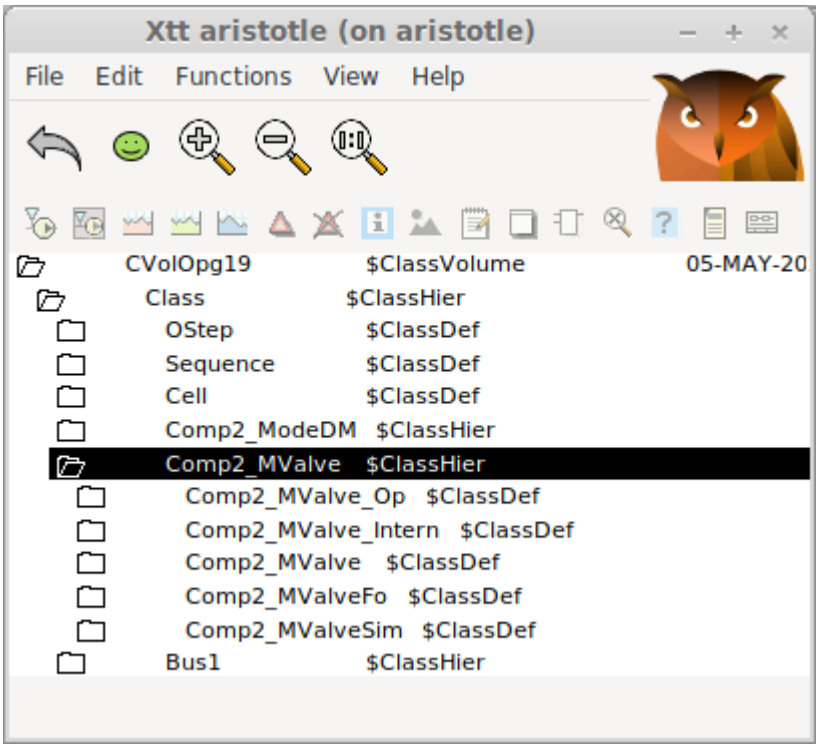


Fig Additional class hierarchy

ClassDef flags Intern, Plc and Obsolete

Three new flags are added to the ClassDef Flags attribute

- Internal Bit for internal classes. When this bit it set, the class will not be displayed in the class volume palette.

- Plc Bit for Plc function objects. These classes will not be displayed in the configurator classvolume palette, only in the plc editor class palette.
- Obsolete. Bit for obsolete classes that will be removed in some later version.

Plc

Bus connections

A bus connection is a convenient way to bundle related signals together in a single entity. It allows you to organize and manage multiple signals as a group. The data structure of the connection is described by a class defined in a class volume. A bus can contain other busses, making it possible to create complex data structures with hierarchical organization.

Beside the bus object a number of function objects can be defined to manage the bus.

SplitBus

Function object to split the bus into it's attributes.

The object has one input for the bus and one output for each attribute in the

bus.

JoinBus

Function object to join a signal to a bus.

The object has one input for each attribute of the bus, and one output for

the bus.

GetBus

Function object to fetch a bus object or bus attribute.

CStoBus

Function object to store a bus into a bus object or bus attribute.

DataToBus

Function object that converts a DataRef to a bus.

The object has an input of type DataRef and an output for the bus.

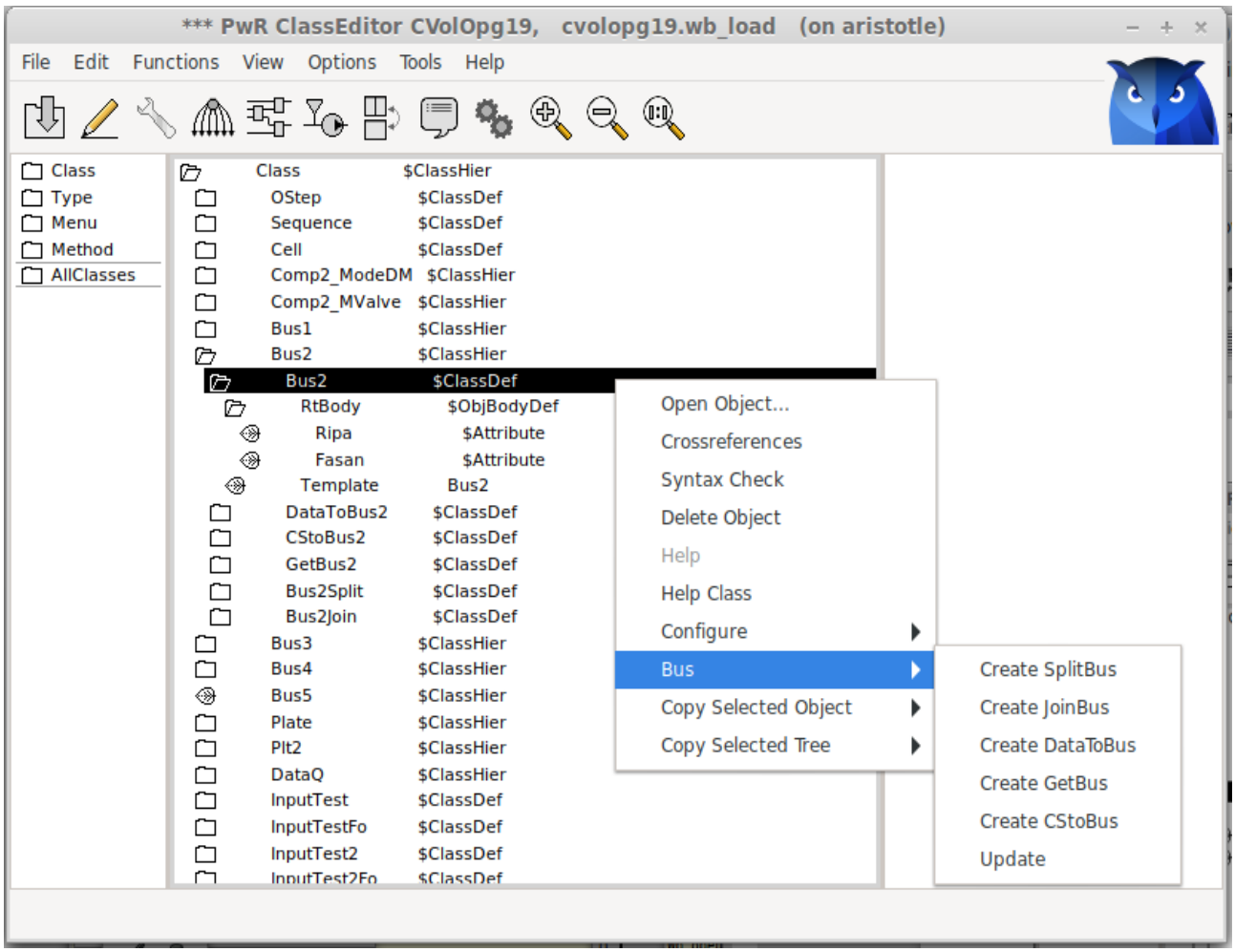


Fig Bus configuration menu

Below is an example of how the SplitBus and JoinBus object are used. The bus contains a sub bus that has it's own split and join methods.

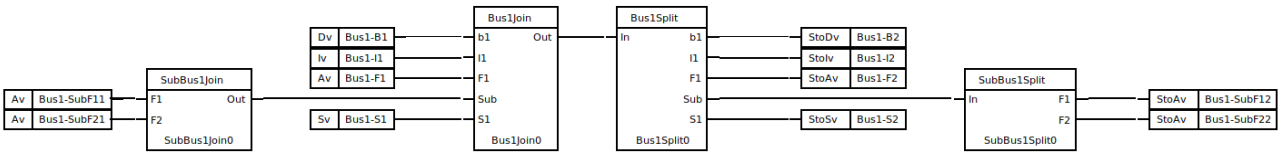


Fig Split and join bus

Inputs and outputs can be defined for any bus connection, and only pins with the same bus definition can be connected. To define an input or output for a specific bus, specify the bus in TypeRef.

Status server communication changed to UDP

The status server previously used SOAP as communication protocol which caused a number of problems, mainly for the supervision center. This is now replaced by UDP. This means that the IP address now has to be part of the configuration for each node in the supervision center. It also means that the supervision center is not compatible with older versions. To maintain the compatibility with older versions of the supervision center, a version of the status server that still

uses SOAP is still available with the name rt_statussrvv59.

Kafka history server

The program rs_export_rtdb sends a selection of values from the real time database to a kafka server.

Configuration

Configuration object

The program is configured with an object of class Ssab_ExportRtdbServer in the node hierarchy. The object is optional, making is possible to build rs_export_rtdb also for older releases.

Kafka configuration file

Kafka is configured with the file $pwrp_load/kafka_config.ini

```
[default]
bootstrap.servers=my.kafka.server.proview.se
ssl.ca.location=/pwrp/common/load/ca23.crt
security.protocol=SASL_SSL
sasl.mechanism=SCRAM-SHA-512
sasl.username=myuser
sasl.password=mypassword
enable.ssl.certificate.verification=True
message.max.bytes=15728640
acks=0
compression.codec=gzip

[consumer]
group.id=my_group
```

Certificate

If certificate verification is needed, a certificate file has to be present and the location for the file specified in the kafka configuration file.

Selection file

The selection file, $pwrp_load/select.json, contains names of the attributes that should be sent to the server. The file is generated by the program rs_export_gen that has to be executed before the server communication can be started. When changes are made in the database, also the selection file has to be updated.

DataQ

DataQ provides a way to store and transport data sets and make sure they are present in the right place at the right time. The coding is made in the plc editor making it possible to follow the data

flow through the system.

Classes

QCreateData

Creates a dynamic object to store the incoming data. The dynamic object is inserted into the connected DataQ object.

DataQ

Container for dynamic (or static) objects. Together with the three types of transport objects it can implement different types of queues and stacks. The DataQ object has order outputs for the first and last object in the queue.

QTrp, QTrpFF, QtrpRR

These are transport objects the transports the data object from one DataQ to another. The transport con be done from rear to front, from front to front, from rear to rear, in forward or backward direction.

QOrder

QOrder objects is connected to the order output of the DataQ objects. The order output is a bus containing the order status, Data object reference and some status flags. Attributes of type Delated, Limited, Condition, Pulse and Stored can be applied on the order.

QRemoteOrder, QTargetOrder

The QRemoteOrder and QTargetOrder pair makes it possible to transfer the order to remote nodes. The QRemotOrder is connected to the order output of a DataQ object. When activated, the activity is transferred to the corresponding QTargetOrder in the remote node. The QTargetOrder will create a local dynamic object in the remote node which is supplied on the order output. The local data object can be a copy or a subset of the original data object, and the conversion is specified in the class volume with $TargetAttributes. When the remote operation is preformed, feedback data is transferred back to the original data object.

RemoteDataQ, TargetDataQ

RemoteDataQ/TargetDataQ is connected to a DataQ object and will handle a sub sequence of DataQ objects. The sub sequence can reside in a remote node. When data objects are entering the main DataQ, the TargetDataQ will also insert a corresponding data object into the sub sequence. During the operations in the sub sequence, feedback data can be sent back to the original data object. When the object reaches the end of the sub sequence, a feedback trigger is set to make it possible for the original data object to leave the main DataQ object.

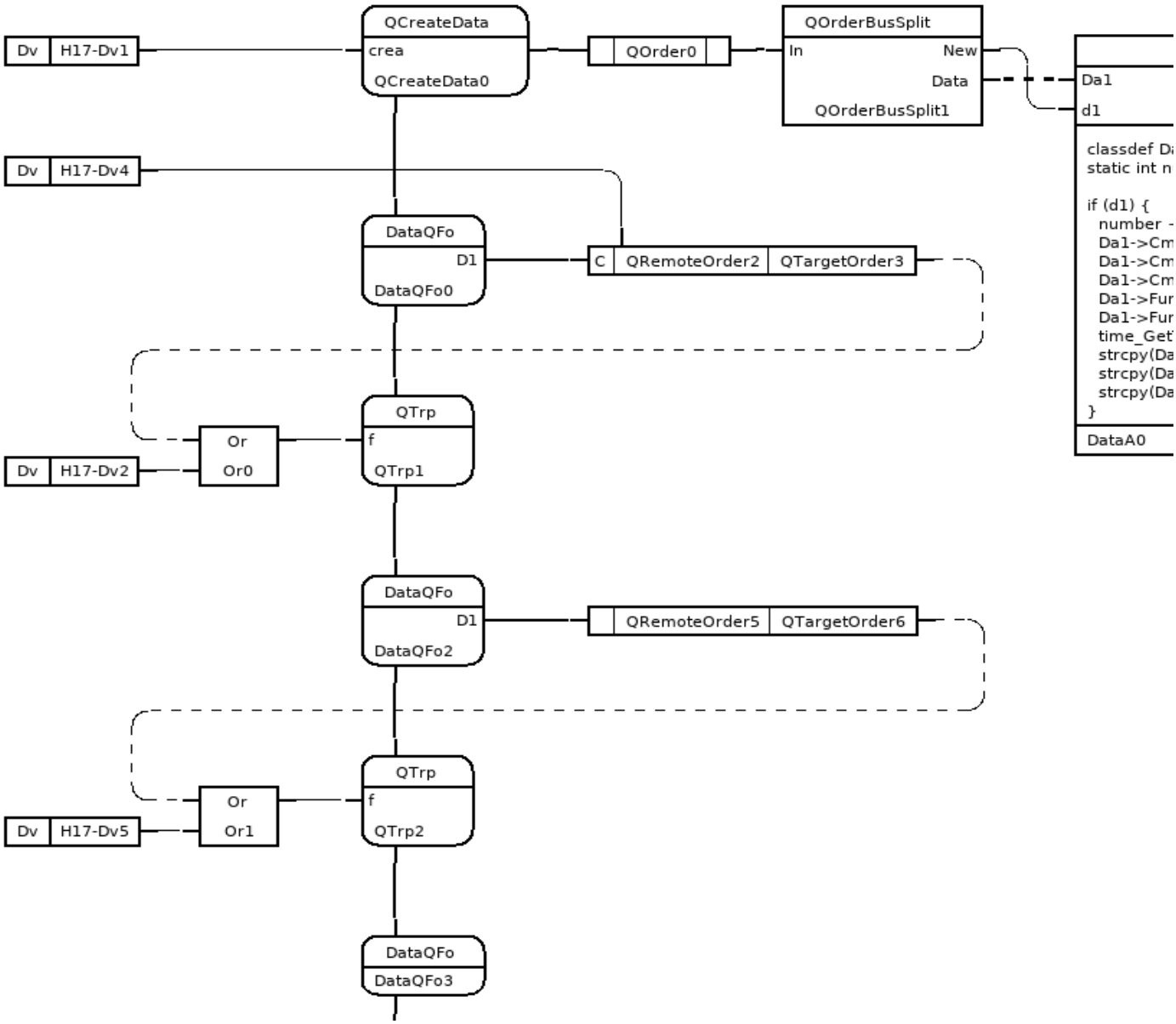


Fig DataQ with QRemoteOrder

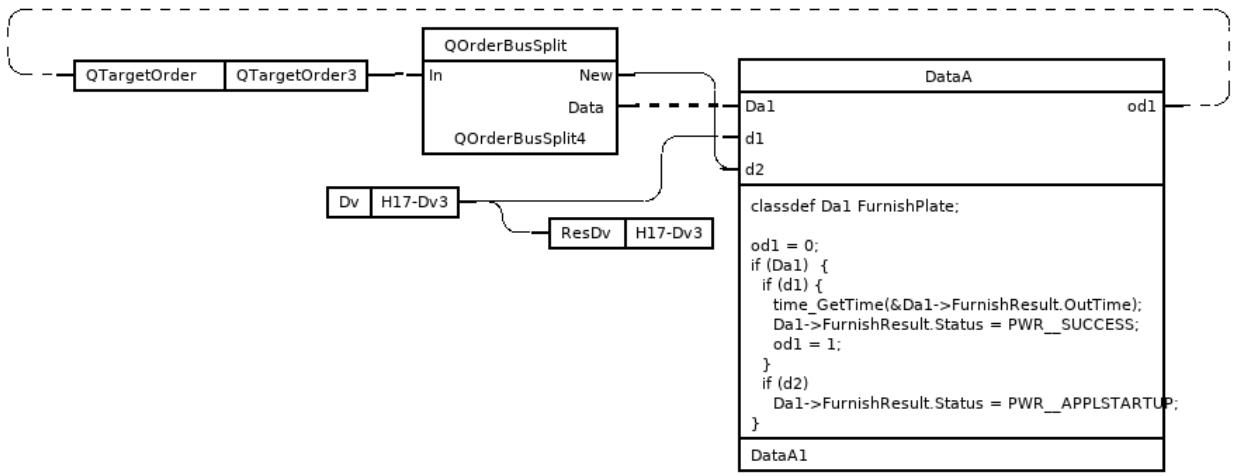


Fig QTargetOrder

TargetAttribute

TargetAttribute is type of attribute that will be linked to a similar attribute in a source object.

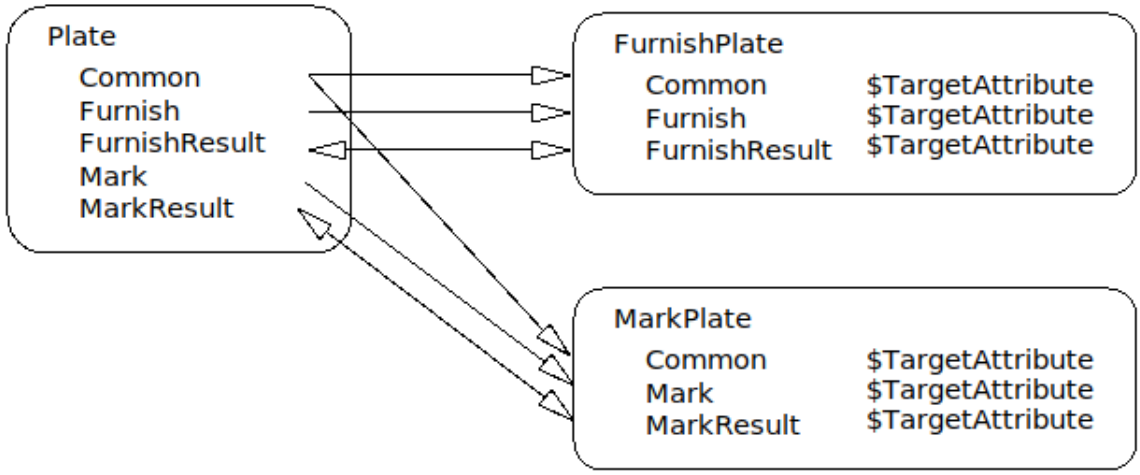


Fig Attribute dependency with TargetAttribute

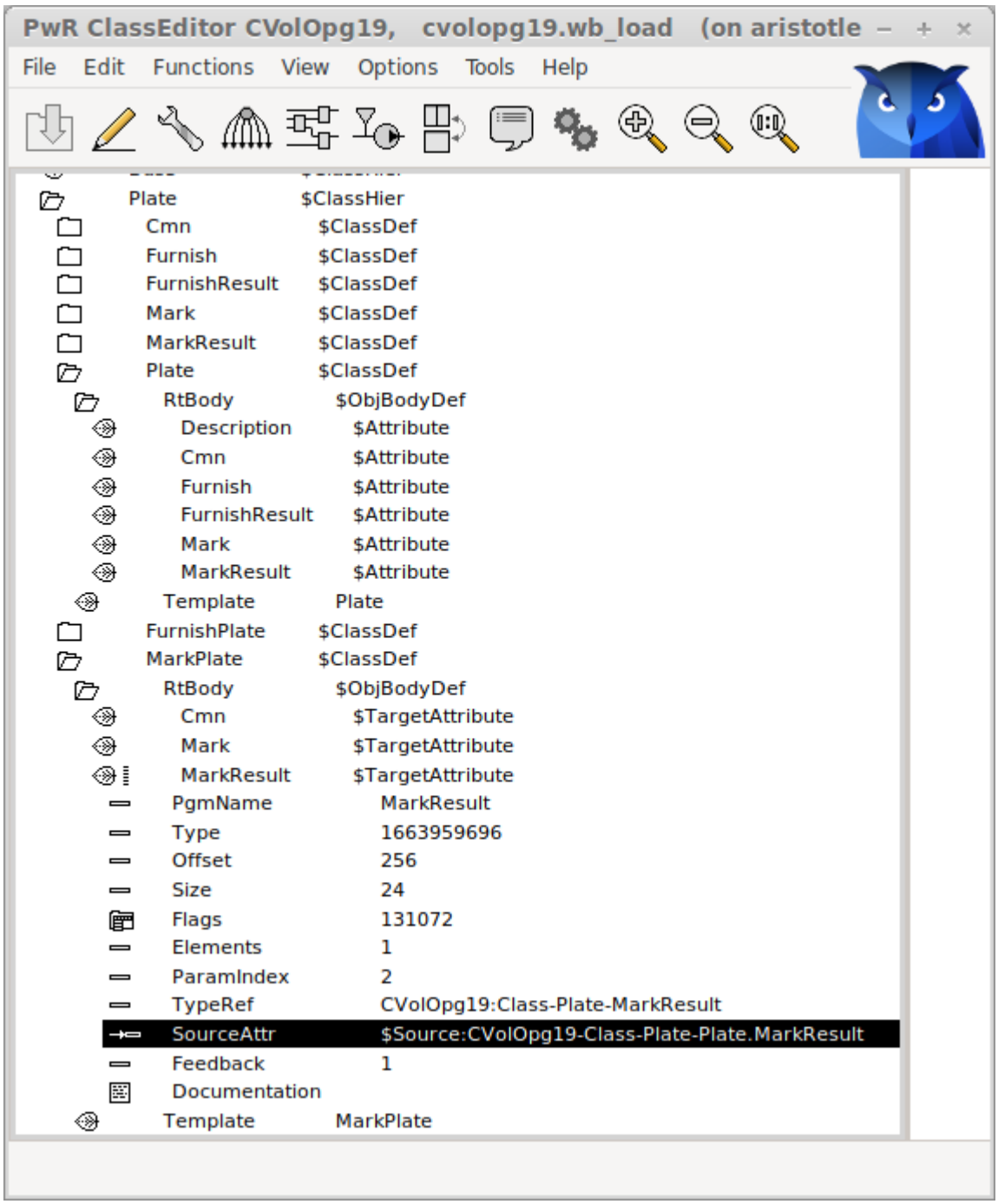


Fig $TargetAttribute.SourceAttr

Support for 64-bit ARM

V6.1 can be built on 64-bit ARM.

Upgrade procedure

The upgrading has to be done from any V6.0. If the project has a lower version, the upgrade has to be performed stepwise following the schema

V2.1 -> V2.7b -> V3.3 -> V3.4b -> V4.0.0 -> V4.1.3
→V4.2.0→V4.5.0→V4.6.0→V4.7.0→V4.8.6→(V5.0.0)→V5.1.0→V5.2.0→V5.3→V5.4→V5.5
→V5.6→V5.7→V5.8→V5.9→V6.0→V6.1

Enter the administrator and change the version of the project to V6.1.0. Save and close the administrator.

Do a sdf to the project and run upgrade.sh.

I you have any class volumes, enter the class editor and build the volume.

Enter the configurator for each root volume and activate 'Function/Update Classes' and build.