Release Notes V5.7

2019 10 28

Table of Contents

Upgrading to ProviewR V5.7.0

This document describes new functions i ProviewR V5.7.0, and how to upgrade a project from V5.6.0 to V5.7.0.

New functions

Qt GUI

The GUI is changed from GTK to Qt with hopefully the same functionality.

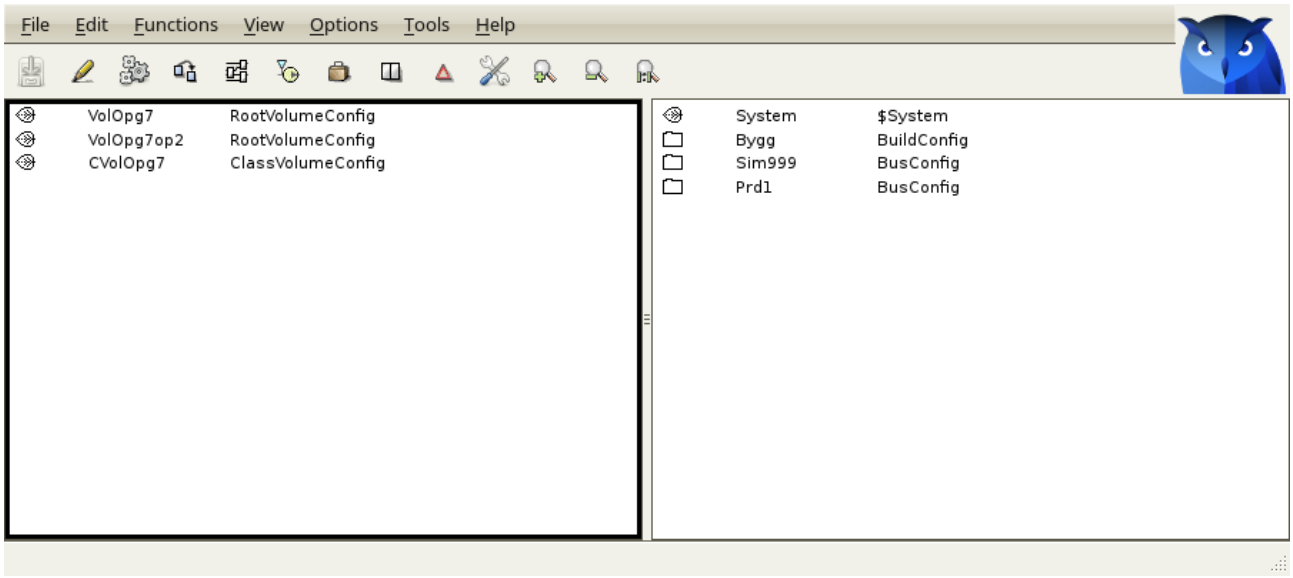


Fig New Qt interface

Python runtime API

The runtime Python API contains a number of classes and functions to get information about the objects in the ProviewR runtime database.

There are function to find objects and investigate their relationship to other objects and to examine the structure and content of the objects.

There are also functions to set up subscriptions to continuously get new updates of attribute values.

The API contains the classes

Oid	object in the runtime database.
Aref	attribute in the runtime database.
Sub	subscription.
Cid	class.
Tid	type.
ADef	attribute definition

To get more information about the classes, enter Python, import pwrrt and type *help(pwrrt.Oid)*, *help(pwrrt.Aref)* etc.

The module is placed on $pwr_exe, that should be included in the PYTHONPATH.
> export PYTHONPATH=$PYTHONPATH:$pwr_exe

Module pwrrt

Functions

application()	Create an application.
attribute()	Get an attribute.
getPriv()	Returns the current privileges.
getSevEvent()	Get events history.
getSevEventsDataFrame()	Get events history data frame..
getSevItemList()	Get history item list.
getSevItemData()	Get history data..
getSevItemsDataFrame()	Get history data frame.
init()	Attach ProviewR runtime.
getUser()	Returns current user.
login()	Login as a ProviewR user.
logout()	Logout from previously logged in user.
object()	Get an object.
subscribe()	Set up an subscription.
volume()	Get volume.
volumes()	Get all volumes.

Class Oid

Runtime object.

attribute()	Get an attribute in the object.
child()	Get first child of the object.
children()	Get all children of the object.
cid()	Get object class.
fullName()	Get the full name of the object.
name()	Get object name.
next()	Get the next sibling of the object.
oidStr()	Get object identity as a string.

parent()	Get the parent of the object.

Class Aref

Object attribute.

arefStr()	Get attribute reference as a string.
fullName()	Get the full name of the attribute.
name()	Get attribute name.
setValue()	Set the value of an attribute.
subscribe()	Set up a subscription of the attribute.
tid()	Get attribute type.
value()	Get attribute value.

Class Sub

A ProvewR subscription.

close()	Close subscription.
setValue()	Set the value of the subscribed attribute.
value()	Get the value of the subscribed attribute.

Class Cid

Object class.

attrObject()	Get the first instance of the class, including attribute objects.
attrObjects()	Get a list with all instances of the class, including attribute objects.
attributes()	Get a list with all attributes defined in the class.
fullName()	Get the full name of the class object.
name()	Get class name.
nextAttrObject()	Get the next instance of the class, including attribute objects.
nextObject()	Get the next instance of the class-
object()	Get the first instance of the class.
objects()	Get a list with all instances of the class.

Class Tid

Attribute type.

fullName()	Get the full name of the type object.
name()	Get type name.

Class Adef

Attribute definition.

cid()	Get the class of the attribute definition object.
elements()	Get number of elements of the attribute.
flags()	Get flags word for the attribute.

name()	Get the name of the attribute.
offset()	Get the offset of the attribute.
size()	Get the size of the attribute.
typeref()	Get the TypeRef of the attribute.

Class ADef

Attribute definition.

cid()	Get class identity for the attribute definition object
elements()	Get number of elements of the attribute.
flags()	Get the flag word of the attribute.
name()	Get the name of the attribute.
offset()	Get the offset of the attribute.
size()	Get the size of the attribute.
typeref()	Get TypeRef of the attribute.

Example 1

Application example.

```
import pwrrt

class Ctx:
    pass

# Open callback function
def appl_open(ctx):
    appl.setStatus('applstartup')
    # Subscribe to some attributes
    ctx.sub1 = pwrrt.subscribe('Demo-Ge-Dynamics-DigLowColor-Dv1.ActualValue')
    ctx.sub2 = pwrrt.subscribe('Demo-Ge-Dynamics-AnalogColor-Av1.ActualValue')
    appl.setStatus('applrun')
    appl.log('info', "Python application up and running !!")

# Scan callback function
def appl_scan(ctx):
    if ctx.sub1.value() == 0:
        ctx.sub2.setValue(22);
    else:
        ctx.sub2.setValue(11);
    print 'Here in scan'

# Close callback function
def appl_close(ctx):
    ctx.sub1 = 0
    ctx.sub2 = 0
    appl.setStatus('none')
    appl.log('info', "Python application closing down")


ctx = Ctx()
appl = pwrrt.application('MyAppl', 'Nodes-DemoNode-Servers-backup', 19, 1,
      appl_open, appl_scan, appl_close, ctx)

appl.mainloop()
```

Example 2

Tkinter example with indicator, value field and pushbutton.

```
from Tkinter import *
import pwrrt


# Button click callback
def button_click_cb():
  a = pwrrt.attribute('Demo-Ge-Dynamics-DigLowColor-Dv1.ActualValue');
  if a.value() == 1:
    a.setValue(0)
  else:
    a.setValue(1)

# Cyclic scan function
def scan():
  global sub1_old
  global sub2_old

  value = sub1.value()
  if value != sub1_old:
    if sub1.value() == 1:
      dv1_label["bg"] = "lightgreen"
    else:
      dv1_label["bg"] = "black"
    sub1_old = value

  value = "%5.2f" % sub2.value()
  if value != sub2_old:
    av1_label["text"] = value
    sub2_old = value

  window.after(500, scan)


pwrrt.init('Example2')
pwrrt.login('pwrp', 'pwrp')

# Create window
window = Tk()
window.title("Proview First Python app ever")
window.geometry('350x200')

# Create button
button = Button(window, text="Toggle Dv1", command=button_click_cb,
bg="lightgray")
button.grid(column=0, row=0, padx=50, pady=50)

# Create indicator label
dv1_label = Label(window, width=3, height=2, bg="black", borderwidth=1,
relief="solid")
dv1_label.grid(column=1, row=0, padx=50, pady=50)

# Create value label
av1_label = Label(window, width=7, bg="white", borderwidth=1, relief="solid",
font=("Helvetica",16))
av1_label.grid(column=1, row=2, padx=20, pady=0)

# Fetch attribute and subscribe
```

```
sub1 = pwrrt.subscribe('Demo-Ge-Dynamics-DigLowColor-Dv1.ActualValue')
sub2 = pwrrt.subscribe('Demo-Ge-Dynamics-AnalogColor-Av1.ActualValue')
sub1_old = -1
sub2_old = ''

scan()
window.mainloop()
```

Python workbench API

The workbench Python API contains a number of classes and functions to get information about the objects in the ProviewR development database.

There are function to find objects and investigate their relationship to other objects and to examine the structure and content of the objects.

There are also functions to build and execute commands.

The API contains the classes

Vid	volume in the development database.
Oid	object in the development database.
Aref	attribute in the development database.
Cid	class.
Tid	type.
ADef	attribute definition.

To get more information about the classes, enter Python, import pwrwb and type help(pwrwb.Oid), help(pwrwb.Aref) etc.

Module pwrwb

Functions

attribute()	Get an attribute.
build()	Build a node.
close()	Close session and database.
closeSession()	Close the current session.
closeWb()	Close database.
command()	Execute a wb command.
getPriv()	Returns the current privileges.
getSessionVolume()	Get the volume of the current session.
getUser()	Returns the current user.
login()	Login as a ProviewR user.
logout()	Logout from the previously logged in user.
object()	Get an object.
open()	Open database and open session to root volume in the database.

openSession()	Open session.
openWb()	Open database.
saveSession()	Save the current session.
sessionIsEmpty()	Returns 1 if no modifications is done since last save, else 0.
version()	Get version.
volume()	Get volume.
volumes()	Get all volumes loaded into the database.

Class Vid

Represents a ProviewR volume.

cid()	Get volume class.
fullName()	Get volume name (same as name()).
name()	Get volume name.
next()	Get next volume.
root()	Get first root object in the volume.
roots()	Get all root objects in the volume.
vidStr()	Get volume identity as a string.

Class Oid

Represents an object in the development database.

attribute()	Get an object attribute.
child()	Get first child of the object.
children()	Get all children of the object.
cid()	Get object class.
fullName()	Get the full name of the object.
name()	Get object name.
next()	Get the next sibling of the object.
oidStr()	Get object identity as a string.
parent()	Get the parent of the object.

Class Aref

Represents an object attribute.

arefStr()	Get attribute reference as a string.
fullName()	Get the full attribute name.
name()	Get attribute name.
setValue()	Set the value of the attribute.
tid()	Get attribute type.
value()	Get attribute value.

Class Cid

Class of a development object.

attrObject()	Get first instance of the class, including attribute objects.
attrObjects()	Get all instances of the class, including attribute objects.
attributes()	Get all attributes defined in the class.
fullName()	Get the full name of the class object.
name()	Get the name of the class.
nextAttrObject()	Get the next instance of the class, including attribute objects.
nextObject()	Get the next instance of the class.
object()	Get the first instance of the class.
objects()	Get all instances of the class.

Class Tid

Attribute type.

fullName()	Get full name of the type object.
name()	Get type name.

Example

An example that finds all the Di objects in a volume.

```
import pwrwb

# Open the database
pwrwb.open('volpwrdemo')

# Attach the root volume
v = pwrwb.volume('volpwrdemo')

# Open a session with write access
pwrwb.openSession(v, 'w')

# Loop through all Di objects
c = pwrwb.Cid('Di')
o = c.object()
while o:
  print o.fullName()
  o = c.nextObject(o)
```

Block objects

A block object is a kind of objects without a class. It’s built with objects of class $Block, $SubBlock and subclasses of $BlockAttribute, such as BlockAttrBoolean, BlockAttrFloat32, BlockAttrInt32 and BlockAttrString. In runtime the block is presented as an object with attributes, and it’s possible to subscribe to the attributes and show them in Ge graphs. Block object can be created at runtime in a dynamic volume, or in the development database.

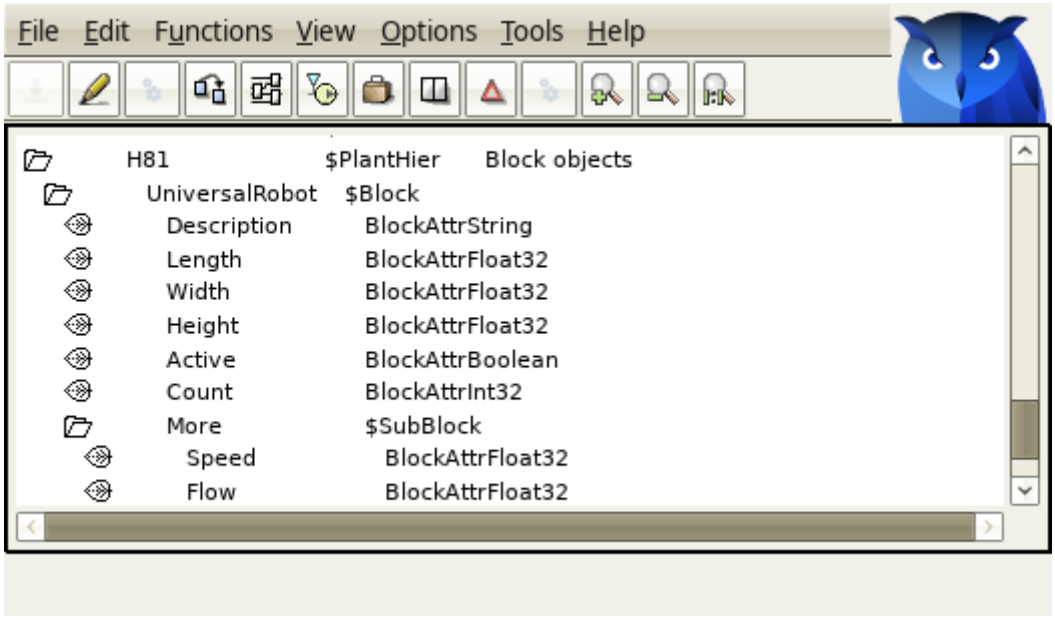


Fig A block object in the development database

In the figure above, $Block is the root of the object and defines the name. The attributes are defined with BlockAttrFloat32, BlocAttrBoolean etc. The $SubBlock defines an object attribute that contains other attribute objects. In the figure below, the object is displayed in the runtime navigator.

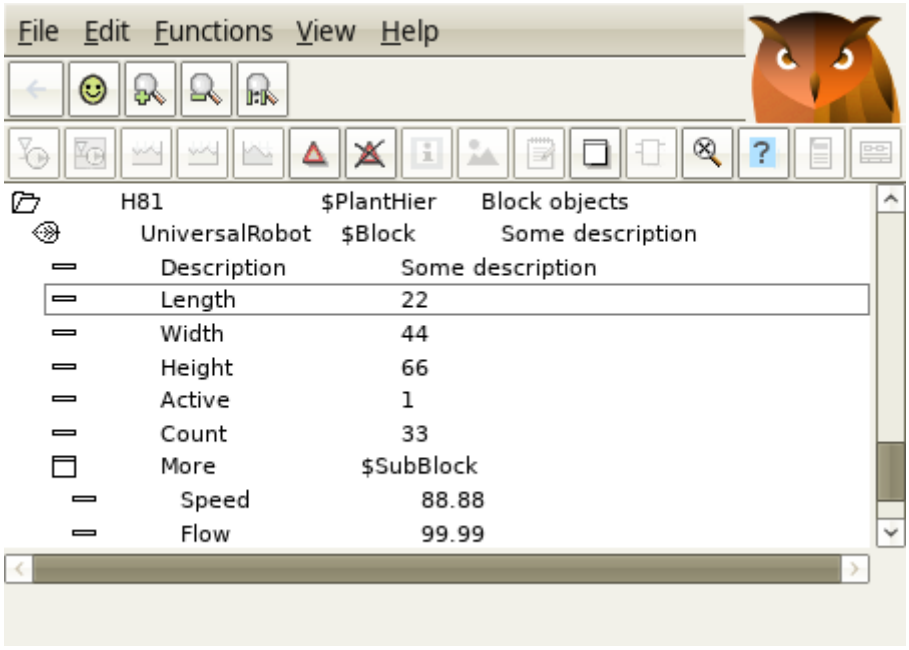


Fig The block object in the runtime navigator

Block objects are used by the Data portal to display stored and exported attributes under the original object name.

Mounting dynamic objects

Dynamic objects can now be mounted with the $MountDynObject object. The mounted object is referenced by string. When the dynamic object is created, a call to gdh_MountDynClients() has to be done to update the mounting.

Data portal

The Data portal is an extended version of the sev server. The idea is that is should serve as a

window to the outer world, and a source for machine learning tools and multivariate analysis. Process data from the whole plant is exported to the portal and available for these tools and also for the new web interface. Python applications can fetch process data, both historical and volatile and make calculations and generate reports.

Still the data portal has a loose connection to the process stations, in order to be able to serve stations of different versions. It's not aware of the hierarchies, identities and classes in the process stations, but tries to recreate the object tree with reverse engineering.

Since V5.6.0 is it possible to start the realtime database in the sev server and to see the current values of the stored attributes under the pwrNode-sev hierarchy. Now it's also possible to export other attributes that doesn't have to be stored but should be available in the realtime database. These attribute values are also present under pwrNode-sev, and created by dynamic object with the new block object feature described above, thus making it possible to display them as the original objects. Further more these dynamics objects can now be mounted in the root volume and form the same object names as in the original nodes. It's then possible to access the exported data from Ge graphs and applications with the original name.

Data export

Data export is configured with SevExport objects. The function is similar to the SevHist objects, but the data is only store in the realtime database in the sev server, not in the history database. As for SevHist objects the data is sent by the sever process rt_sevhistmon, and a SevHistThread is specified in the SevExport object defining the scantime.

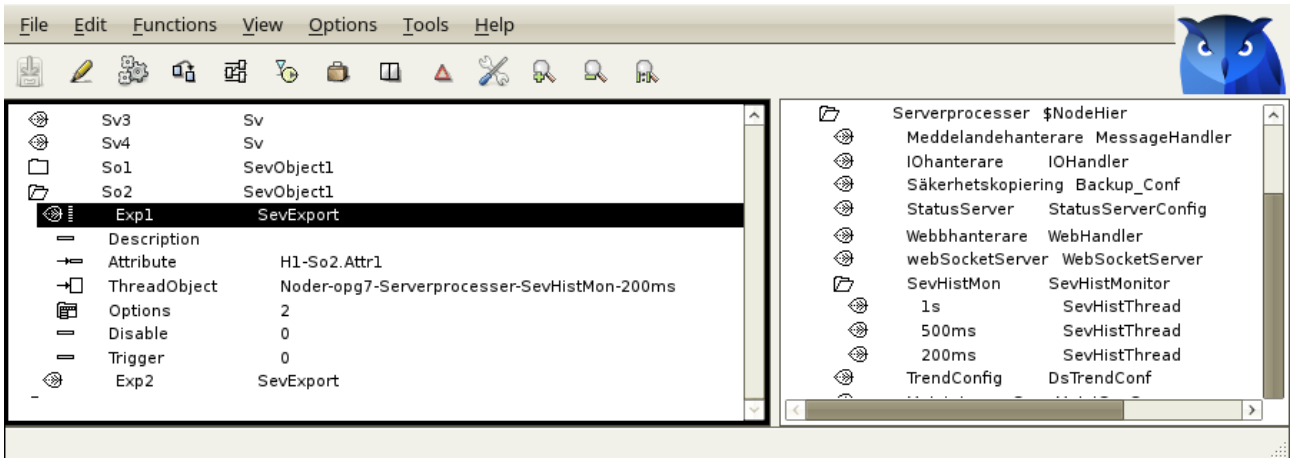


Fig Export client configuration

In the server, the export is configured with a SevImportServer object in the node hierarchy.

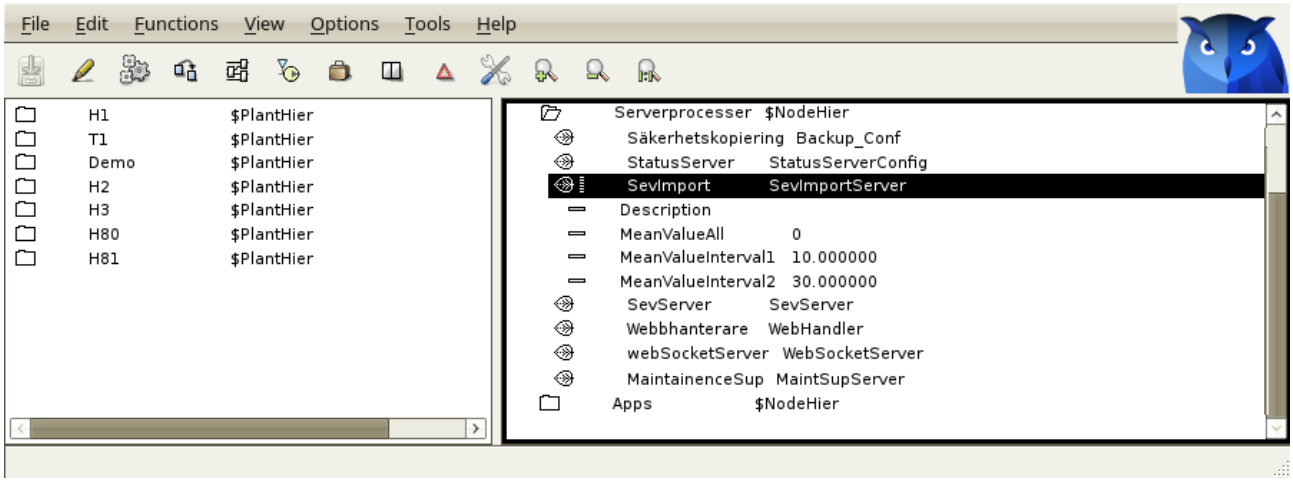


Fig Export server configuration

Mounting

The data exported to the sev_server, both export data and history data, are stored in dynamic objects in the system volume under pwrNode-sev. In V5.7.0 these objects are created with Block objects, described above, thus recreating the structure of the original object. These object can be mounted in the plant hierarchy with $MountDynObject, and in this way appear under the original path.

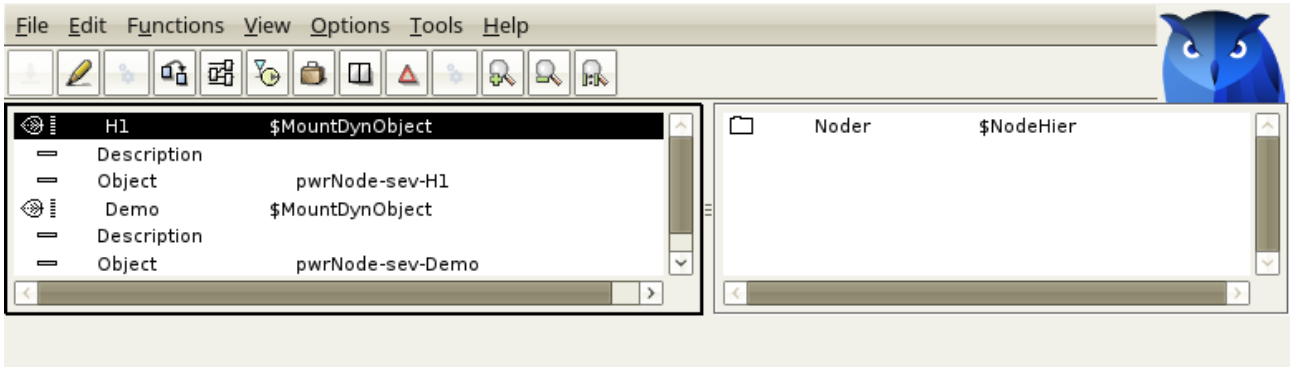


Fig Mount object in the configurator

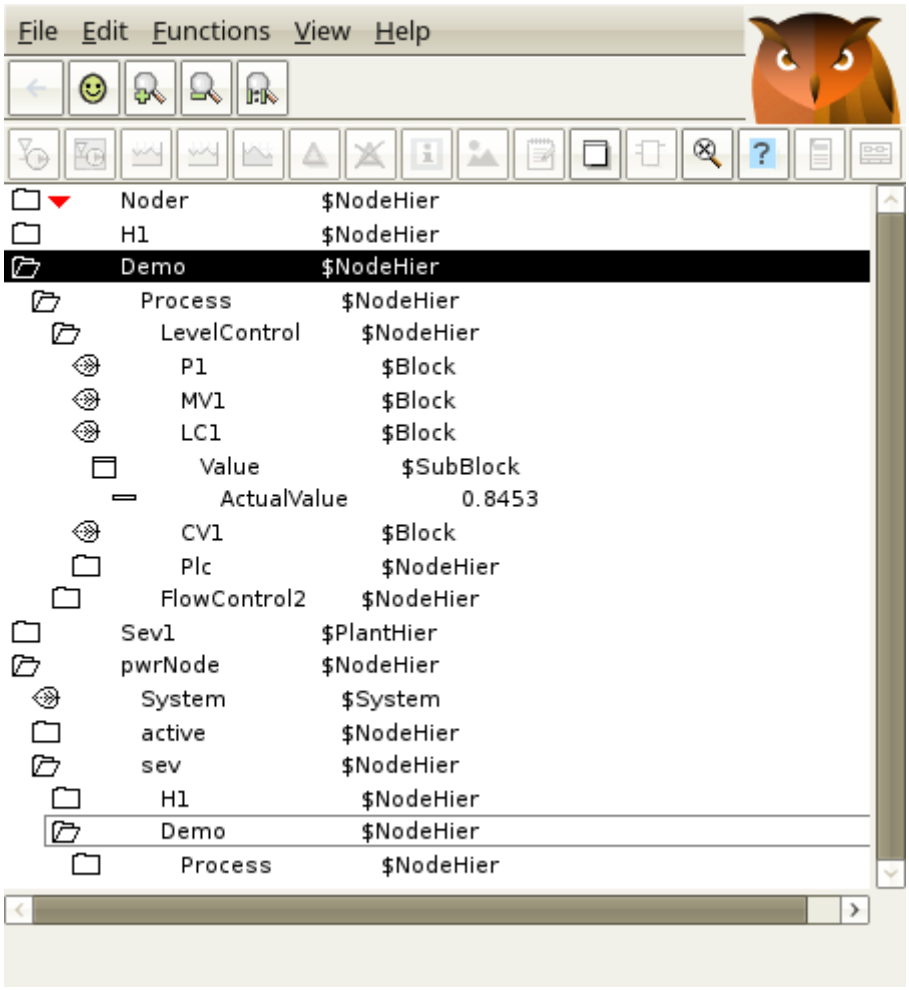


Fig The mount in runtime

Multivariate analyzer

With multivariate analyzer it is possible to view and analyze process history data and logged data. It is also possible to linearize and transform the data and apply machine learning tools as linear regression and neural networks that can be used in models and MPC controllers.

Dataset

A dataset contains data ordered in columns and rows. The first column is the sample time, and the next columns contains measured data for process variables. The data can be fetch from a sev server, generated by the Xtt logging function or read from a cvs-file.

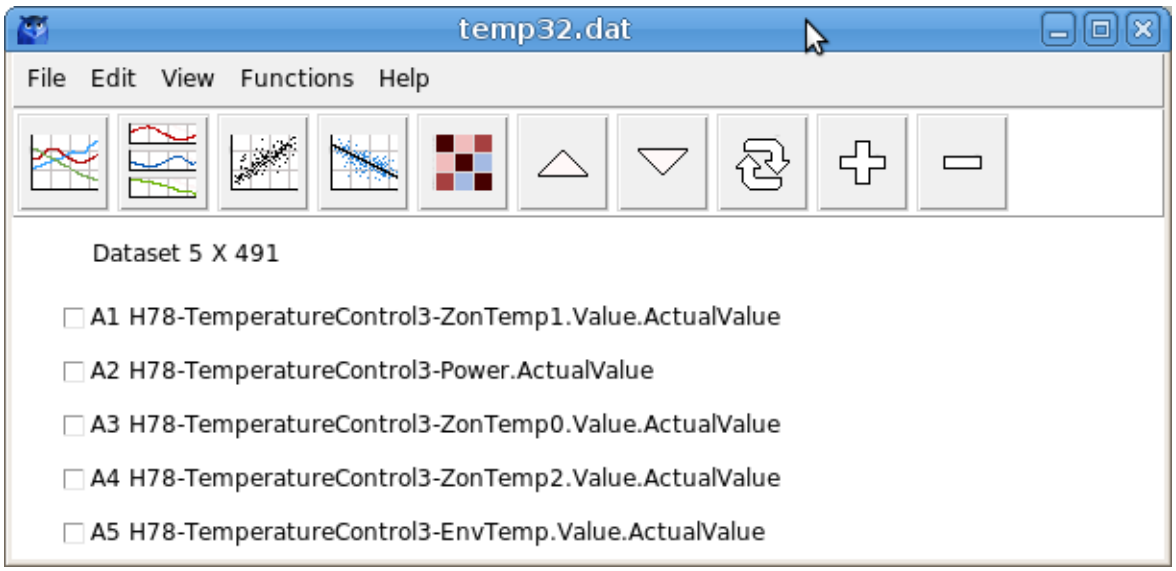


Fig Dataset

Sev server

Data is fetch from a sev sever from *File/Import from server* in the menu. The server host name and an optional item filter is supplied. The items that matches the filter are then displayed and items that should be part of the dataset can be selected. Finally start and end time is entered and the data is fetched and inserted into the dataset.

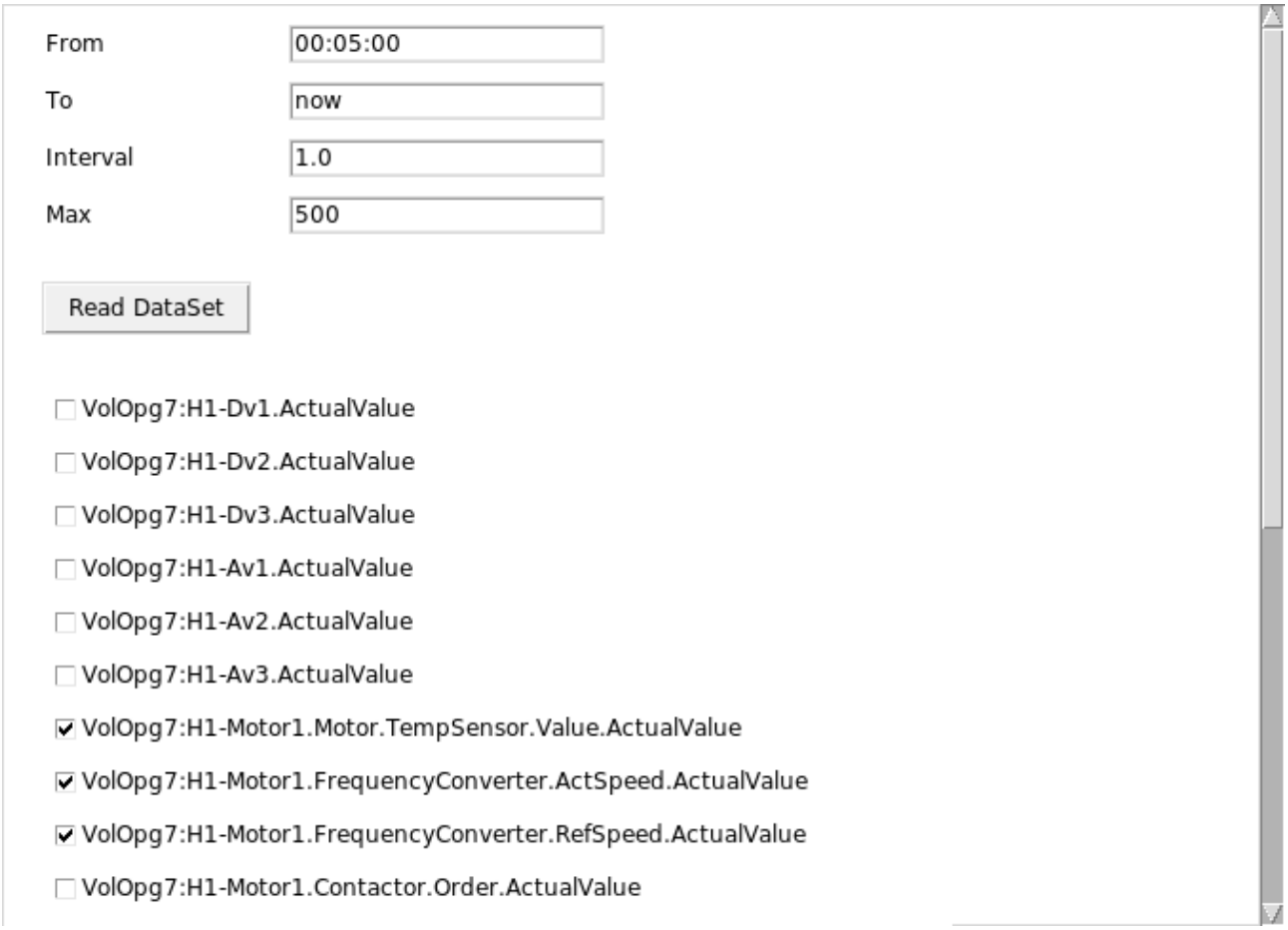


Fig Fetch data from sev server

Xtt logging

Parameters are collected and inserted into a logging entry, and to get the correct time format, 'Format' is set to 1. When the logging is executed, the analyzer can be opened from the 'Analyze' button in the logging entry.

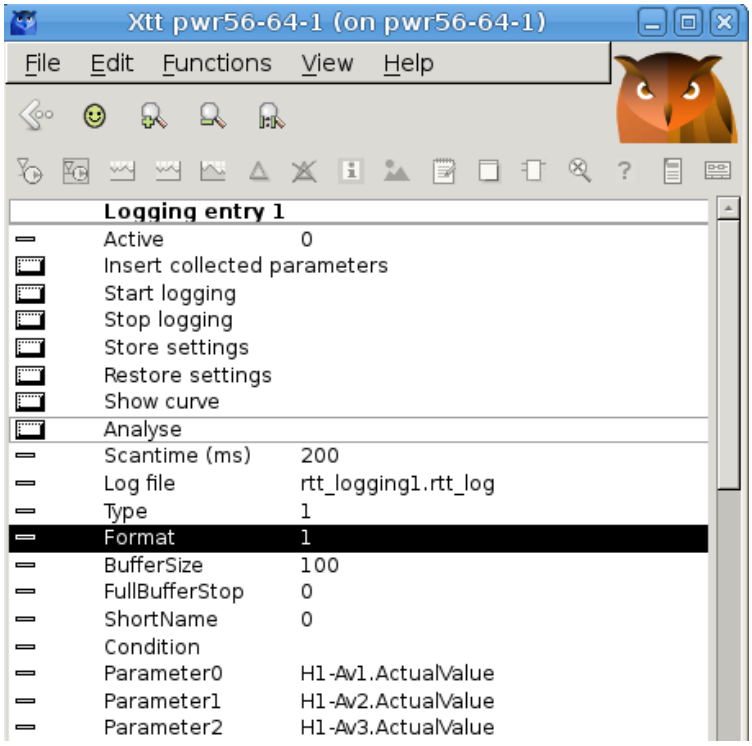


Fig Xtt logging

cvs file

Data can be read from cvs files with format displayed below. The first row a header row with ‘Time’ and the name of each parameter. The next rows contains the time and the parameter values at this time. The file is opened from File/Open in the menu.

```
Time,H78-TemperatureControl3-ZonTemp1.Value.ActualValue, H78-TemperatureControl3-
Power.ActualValue, H78-TemperatureControl3-ZonTemp0.Value.ActualValue, H78-Tempera
tureControl3-ZonTemp2.Value.ActualValue, H78-TemperatureControl3-EnvTemp.Value.Ac
tualValue
2019-05-13 09:25:16.11, 130.500320, 0.000032, 190.000000, 180.000000, 21.500000
2019-05-13 09:25:16.62, 130.500320, 0.000032, 190.000000, 180.000000, 21.500000
2019-05-13 09:25:17.12, 130.500320, 0.000032, 190.000000, 180.000000, 21.500000
2019-05-13 09:25:17.62, 130.500320, 0.000032, 190.000000, 180.000000, 21.500000
2019-05-13 09:25:18.12, 130.500320, 0.000032, 190.000000, 180.000000, 21.500000
...
```

Plots

A number of different plots can be made, for example scatterplot that shows the relationship between two columns, or correlation heatmap that displays the correlation between columns with colors. Dark red is high correlation and dark blue high negative correlation while light tones are low correlation.

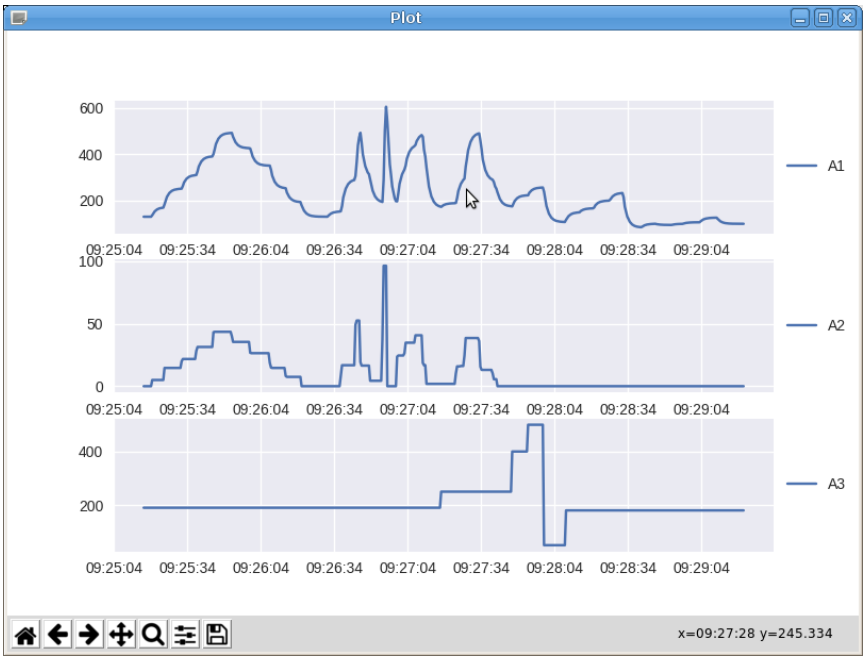


Fig Plot

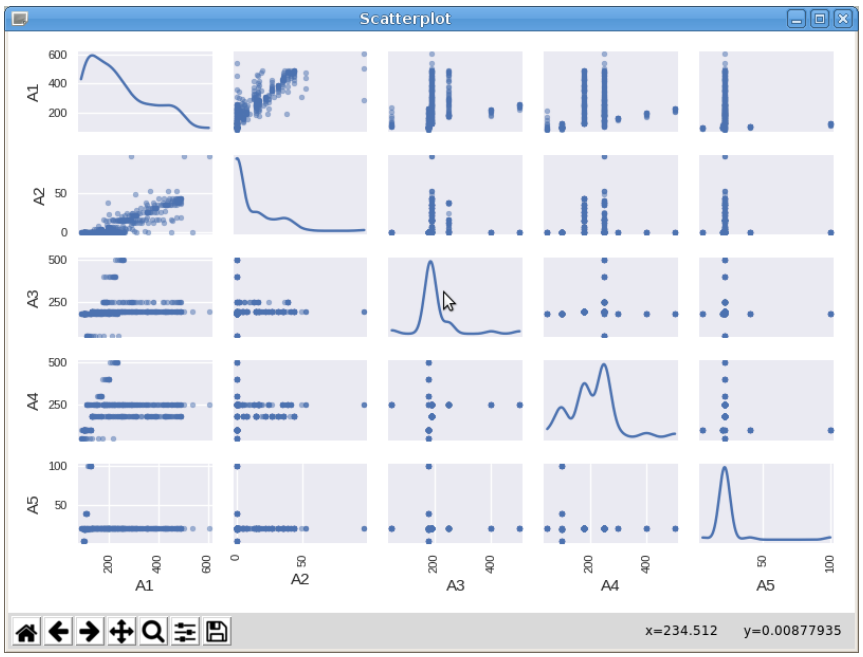


Fig Scatterplot

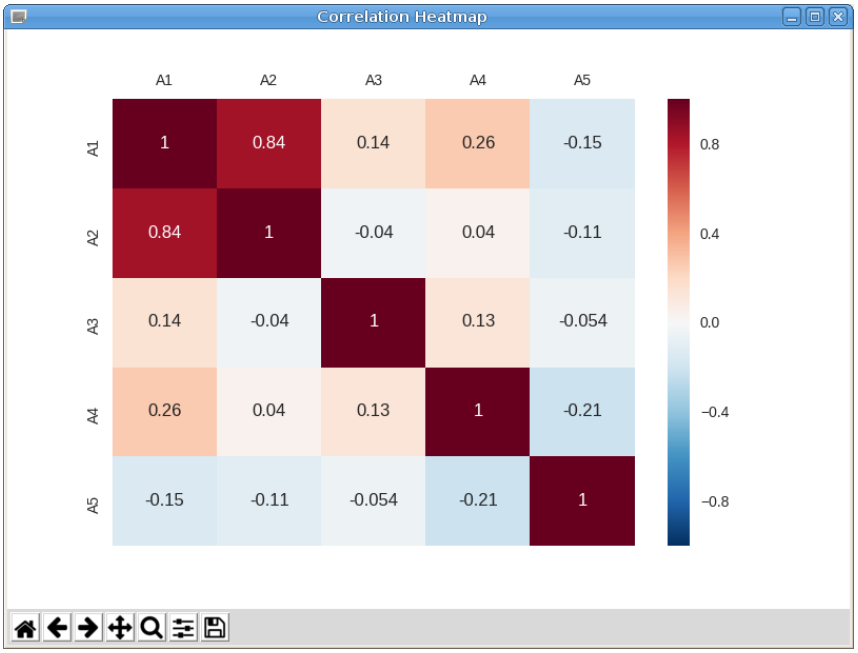


Fig Correlation heatmap

Edit data

The dataset can be edit with a number of functions

- Split will split the dataset into two datasets.
- Clip will pick out a portion of the dataset.
- Join will concatenate two datasets.
- Multiply will create a dataset where the current set is repeated a number of times.
- Move up and down will change the order of the columns.

Transform data

Creating models with Linear regression requires that the columns in the dataset have linear

dependencies. Often this is not the case. The level in a cylinder tank for example has not a linear relationship to the in and out flow, but to the integral of the in and out flow. After an integration of the flow columns there will be a linear relationship and the linear regression can be performed.

Convert column

There are a number of functions to transform the data of a column

- Norm. Not yet implemented.

- Square. Calculate the square of each row.

- Squareroot. Take the square root of each value.

- Exp. Exponential function.

- Log. Logarithmic function.

- Integral. Time integral.

- Derivate. Time derivate.

- Curve. Linear interpolation from a table specified in a cvs file with data points, eg

```
0,0
30,10
70,90
100,100
```

- Shift. Values in the column will be shifted forward or backward. The number of positions the values will be shifted are specified. Positive value will shift forwards and negative backwards.

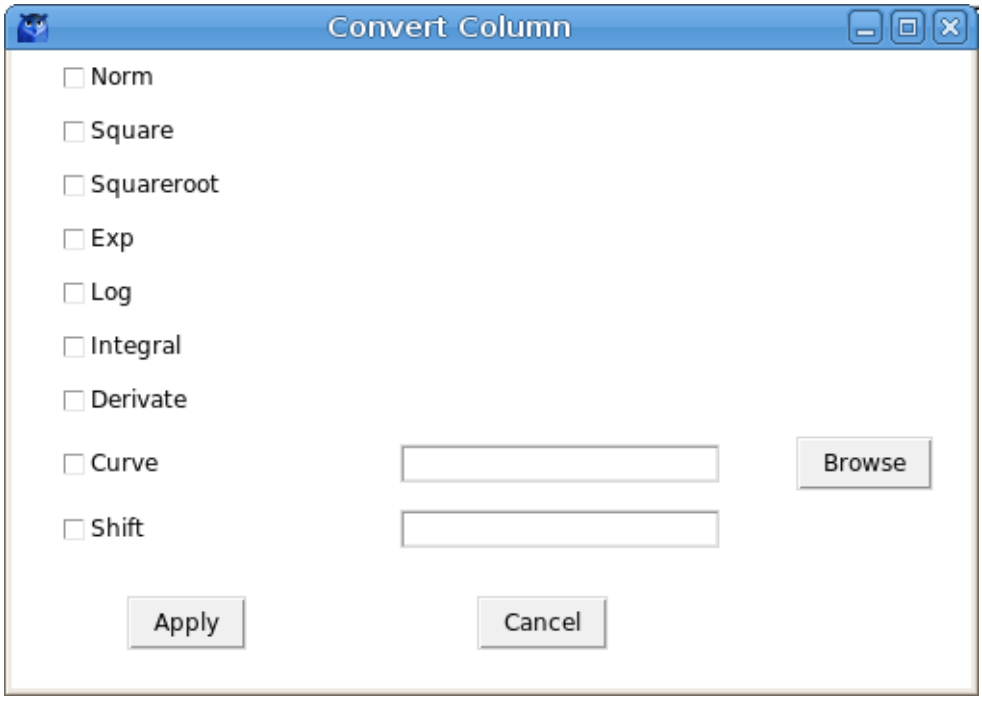


Fig Convert column alternatives

Add column

Add column will in most cases transform the data of one or two columns and put the transformed data in a new column.

- Copy. Make a copy of the selected column.

- Norm. Not yet implemented.

- Square. Calculate the square of each row.

- Squareroot. Take the square root of each value.

- Exp. Exponential function.

- Log. Logarithmic function.

- Integral. Time integral.

- Derivate. Time derivate.

- Add. Add the two selected columns.

- Sub. Subtract between two selected columns. The order of the columns in the dataset is of importance here. The lower positioned column will be subtracted from the higher positioned column.

- Multiply. Multiply the two selected columns.

- Divide. Division of the two selected columns. The higher positioned columns will be divided by the lower positioned.

- Curve. Linear interpolation from a table specified in a cvs file.

- Constant. Will create a column where all rows has the specified value.

- Shift. Values in the column will be shifted forward or backward.

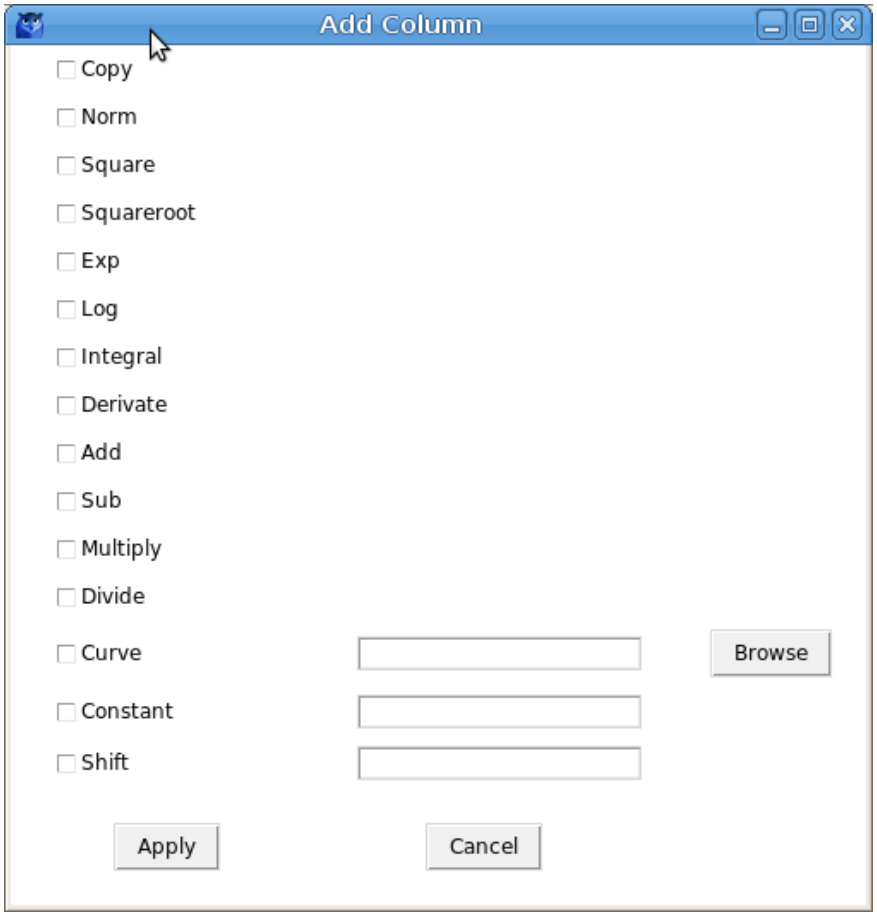


Fig Add column alternatives

Formula

The transformation of a dataset can contain several steps, and when the transformation is finished, the sequence can be stored as a formula and then be applied on other samplings of the same parameters. The saving is made from *File/Save Formula* in the menu, and applied from *File/Apply*

Formula.

Linear regression

Linear regression will create a model where one parameter, y, can be calculated from a number of input values x_1 - x_n . y is supposed to have linear dependencies of the input values, and the formula is

$$y = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_3 + \ldots + a_n x_n$$

where a_0 - a_n will be calculated.

If the dependencies are not linear they first have to be linearized with the transformation tools described above. When the model is used in runtime, the process values have to go through the same transformation before they are used in the regression model.

Lasso and Rigde regression are variants of linear regression that is also implemented.

Fig Linear regression

MLP regressor

MLP (Multi Level Perceptron) is a neural network with an input layer, a number of hidden layers and an output layer. Each node in the hidden and output layers is a neuron that uses a nonlinear activation function. The MLP uses a learning technique called backpropagation.

Before the the training can start, setting for the MLP like number of hidden layers and layer sizes, activation function etc has to be set.

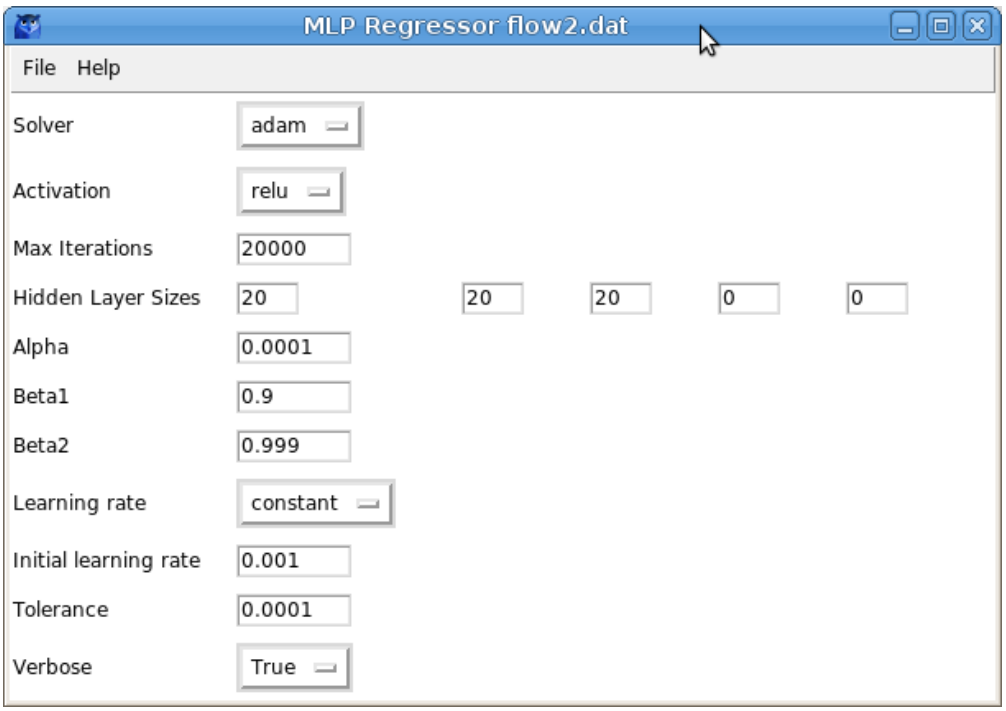


Fig MLP regressor settings

From *File/Create Model* in the menu, the training is started and when it's finished the score is displayed and the model values are plotted with the process values. The module can be written to file with File/Export Model and then used by a MPC controller or model object.

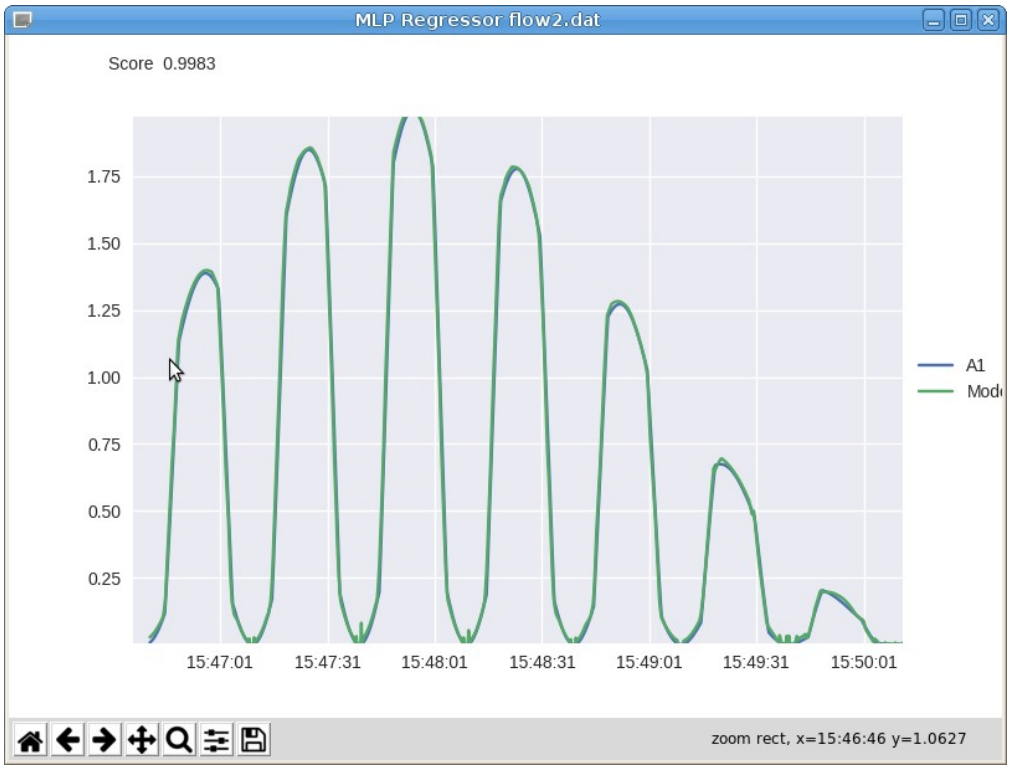
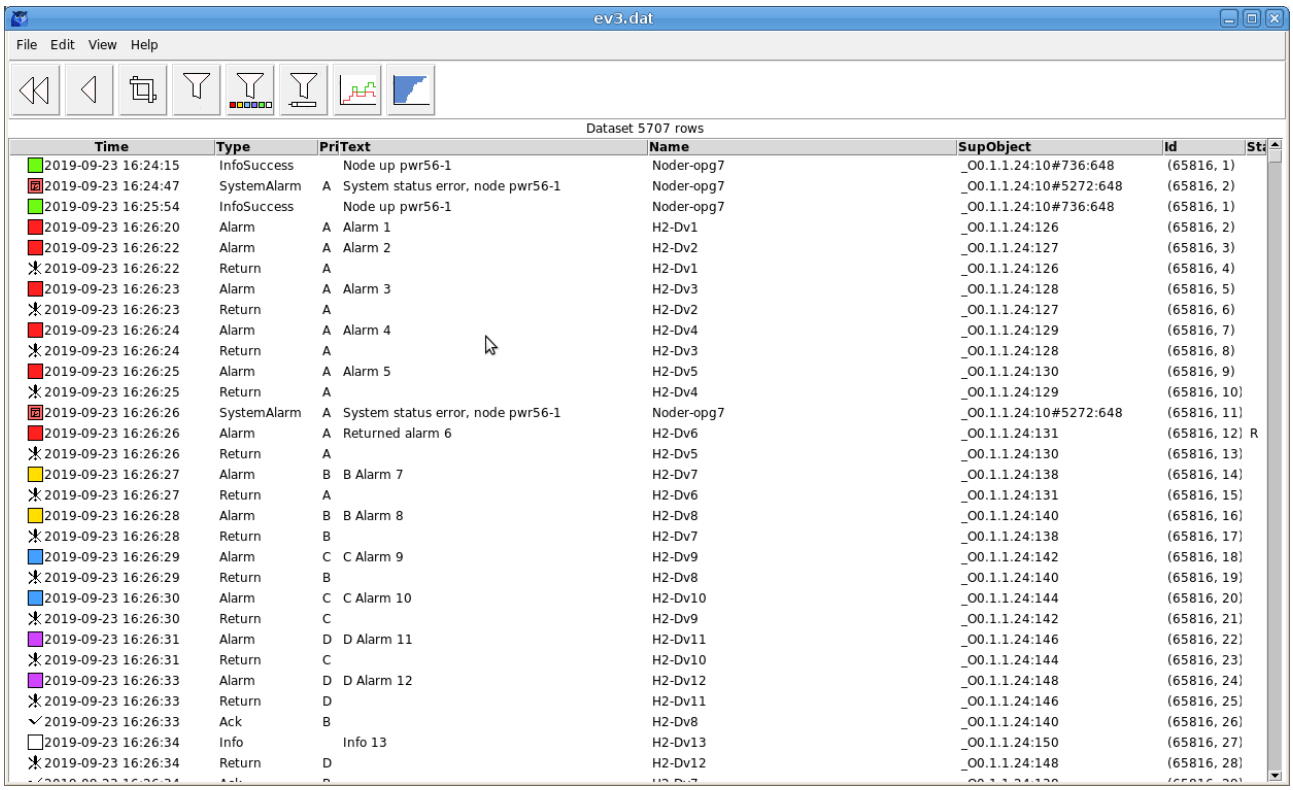


Fig Traning result

Alarm and event analyzer

The alarm and event analyzer can fetch alarm from the sev server, the eventlog or eventlist, and display statistics and plots over the alarm situation. A number of filter functions are available to pick out event of a specific type or priority, or show event from a specific sup object.



Time	Type	Pri	Text	Name	SupObject	Id	Sta
2019-09-23 16:24:15	InfoSuccess		Node up pwr56-1	Noder-opg7	_O0.1.1.24:10#736:648	(65816, 1)	
2019-09-23 16:24:47	SystemAlarm	A	System status error, node pwr56-1	Noder-opg7	_O0.1.1.24:10#5272:648	(65816, 2)	
2019-09-23 16:25:54	InfoSuccess		Node up pwr56-1	Noder-opg7	_O0.1.1.24:10#736:648	(65816, 1)	
2019-09-23 16:26:20	Alarm	A	Alarm 1	H2-Dv1	_O0.1.1.24:126	(65816, 2)	
2019-09-23 16:26:22	Alarm	A	Alarm 2	H2-Dv2	_O0.1.1.24:127	(65816, 3)	
2019-09-23 16:26:22	Return	A		H2-Dv1	_O0.1.1.24:126	(65816, 4)	
2019-09-23 16:26:23	Alarm	A	Alarm 3	H2-Dv3	_O0.1.1.24:128	(65816, 5)	
2019-09-23 16:26:23	Return	A		H2-Dv2	_O0.1.1.24:127	(65816, 6)	
2019-09-23 16:26:24	Alarm	A	Alarm 4	H2-Dv4	_O0.1.1.24:129	(65816, 7)	
2019-09-23 16:26:24	Return	A		H2-Dv3	_O0.1.1.24:128	(65816, 8)	
2019-09-23 16:26:25	Alarm	A	Alarm 5	H2-Dv5	_O0.1.1.24:130	(65816, 9)	
2019-09-23 16:26:25	Return	A		H2-Dv4	_O0.1.1.24:129	(65816, 10)	
2019-09-23 16:26:26	SystemAlarm	A	System status error, node pwr56-1	Noder-opg7	_O0.1.1.24:10#5272:648	(65816, 11)	
2019-09-23 16:26:26	Alarm	A	Returned alarm 6	H2-Dv6	_O0.1.1.24:131	(65816, 12)	R
2019-09-23 16:26:26	Return	A		H2-Dv5	_O0.1.1.24:130	(65816, 13)	
2019-09-23 16:26:27	Alarm	B	B Alarm 7	H2-Dv7	_O0.1.1.24:138	(65816, 14)	
2019-09-23 16:26:27	Return	A		H2-Dv6	_O0.1.1.24:131	(65816, 15)	
2019-09-23 16:26:28	Alarm	B	B Alarm 8	H2-Dv8	_O0.1.1.24:140	(65816, 16)	
2019-09-23 16:26:28	Return	B		H2-Dv7	_O0.1.1.24:138	(65816, 17)	
2019-09-23 16:26:29	Alarm	C	C Alarm 9	H2-Dv9	_O0.1.1.24:142	(65816, 18)	
2019-09-23 16:26:29	Return	B		H2-Dv8	_O0.1.1.24:140	(65816, 19)	
2019-09-23 16:26:30	Alarm	C	C Alarm 10	H2-Dv10	_O0.1.1.24:144	(65816, 20)	
2019-09-23 16:26:30	Return	C		H2-Dv9	_O0.1.1.24:142	(65816, 21)	
2019-09-23 16:26:31	Alarm	D	D Alarm 11	H2-Dv11	_O0.1.1.24:146	(65816, 22)	
2019-09-23 16:26:31	Return	C		H2-Dv10	_O0.1.1.24:144	(65816, 23)	
2019-09-23 16:26:33	Alarm	D	D Alarm 12	H2-Dv12	_O0.1.1.24:148	(65816, 24)	
2019-09-23 16:26:33	Return	D		H2-Dv11	_O0.1.1.24:146	(65816, 25)	
2019-09-23 16:26:33	Ack	B		H2-Dv8	_O0.1.1.24:140	(65816, 26)	
2019-09-23 16:26:34	Info		Info 13	H2-Dv13	_O0.1.1.24:150	(65816, 27)	
2019-09-23 16:26:34	Return	D		H2-Dv12	_O0.1.1.24:148	(65816, 28)	

Fig Alarm and event analyzer

Some plots are the *Event frequency histogram* that show the most frequent alarms, and *Not returned alarms* that show the number for concurrent alarms as a function time.

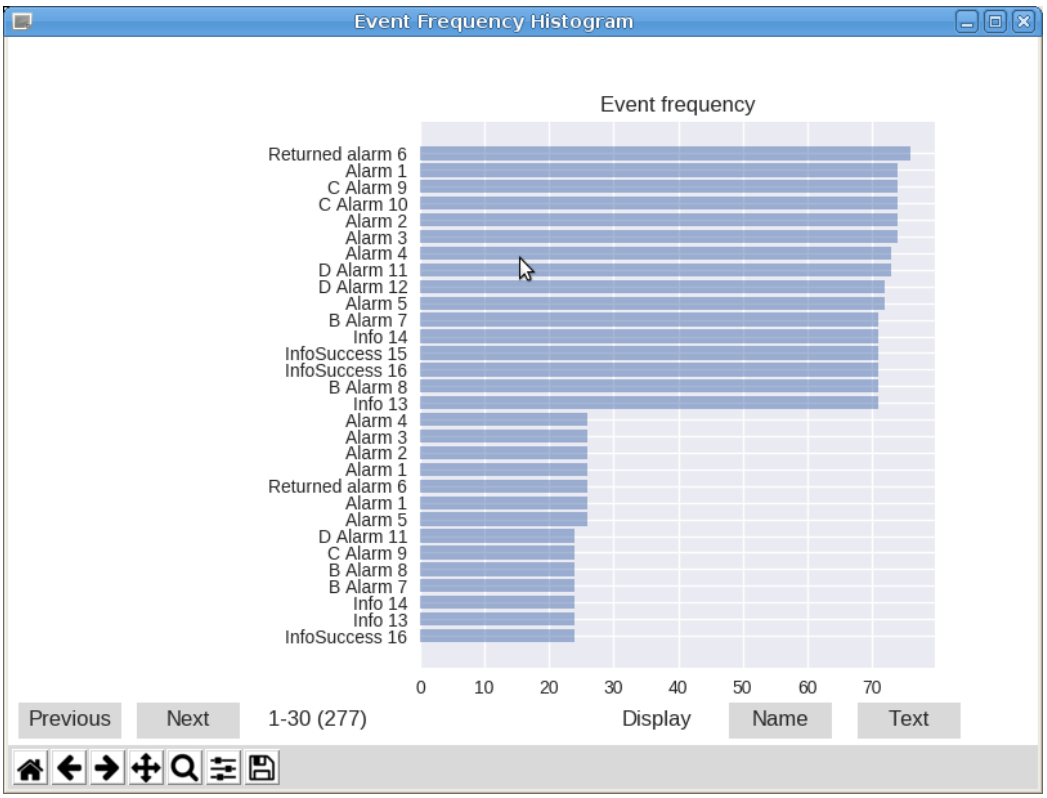


Fig Event frequency histogram

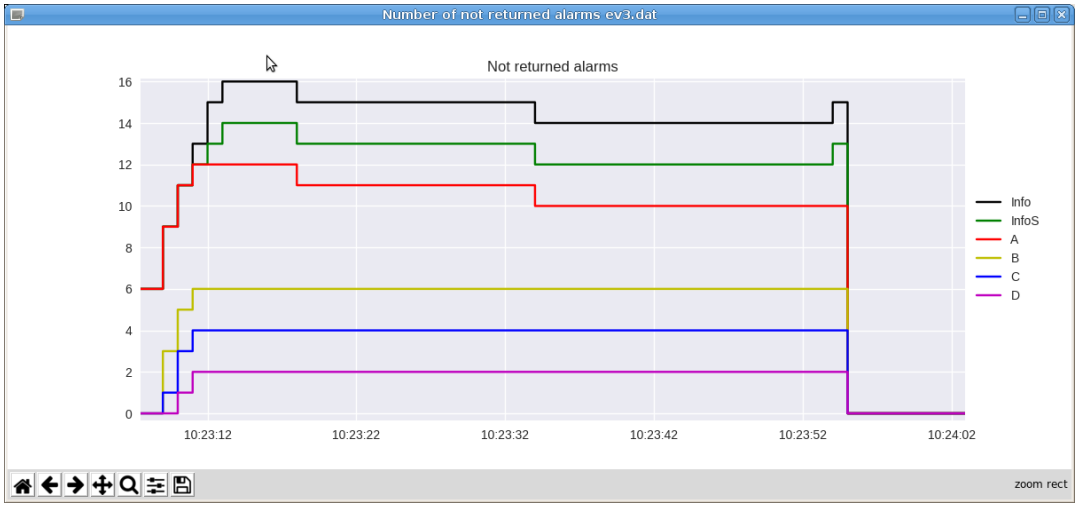


Fig Not return alarms

Simulation objects

Sim_SignalGenerator

Generates a signal value of shape sine, halfsine, square or sawtooth. Noise or exponential filter can be applied to the signal.

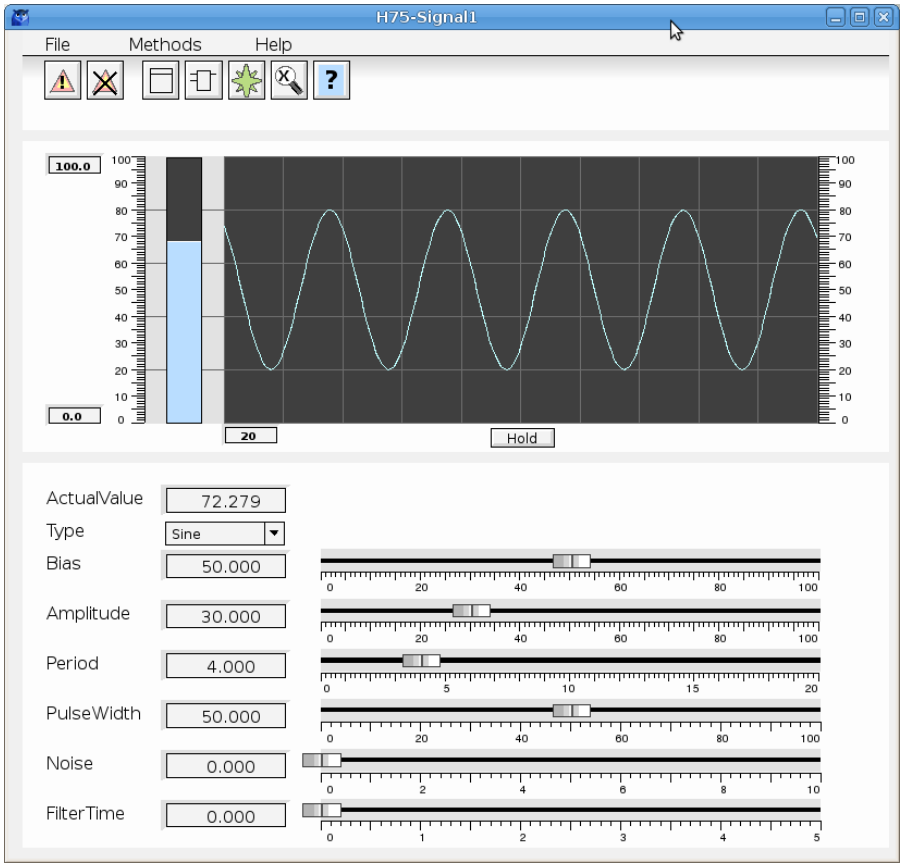


Fig Sim_SignalGenerator object graph

The class is divided in a main object and a function object.

Sim_CylinderTank

Simulates the fluid level in a tank of cylinder shape. Input and output flow are inputs to the object.

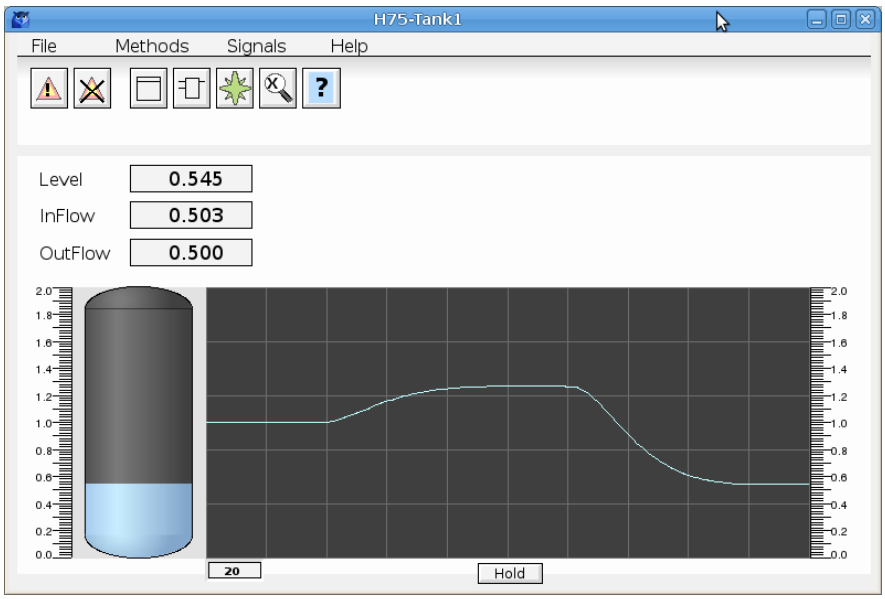


Fig Sim_CylinderTank object graph

Sim_Furnace

Simulates a furnace with heating power as input.

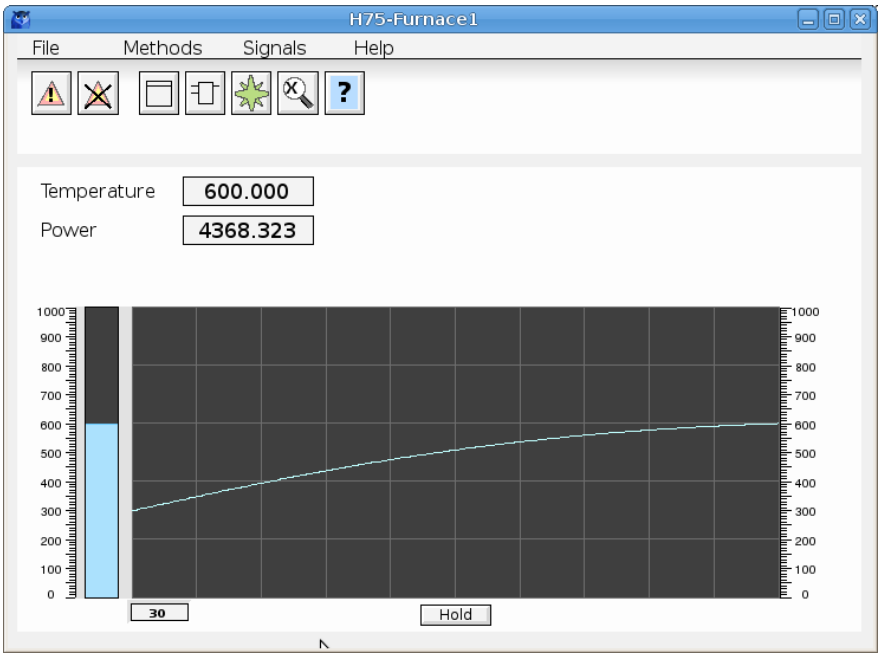


Fig Sim_Furnace object graph

Sim_ModelMLP

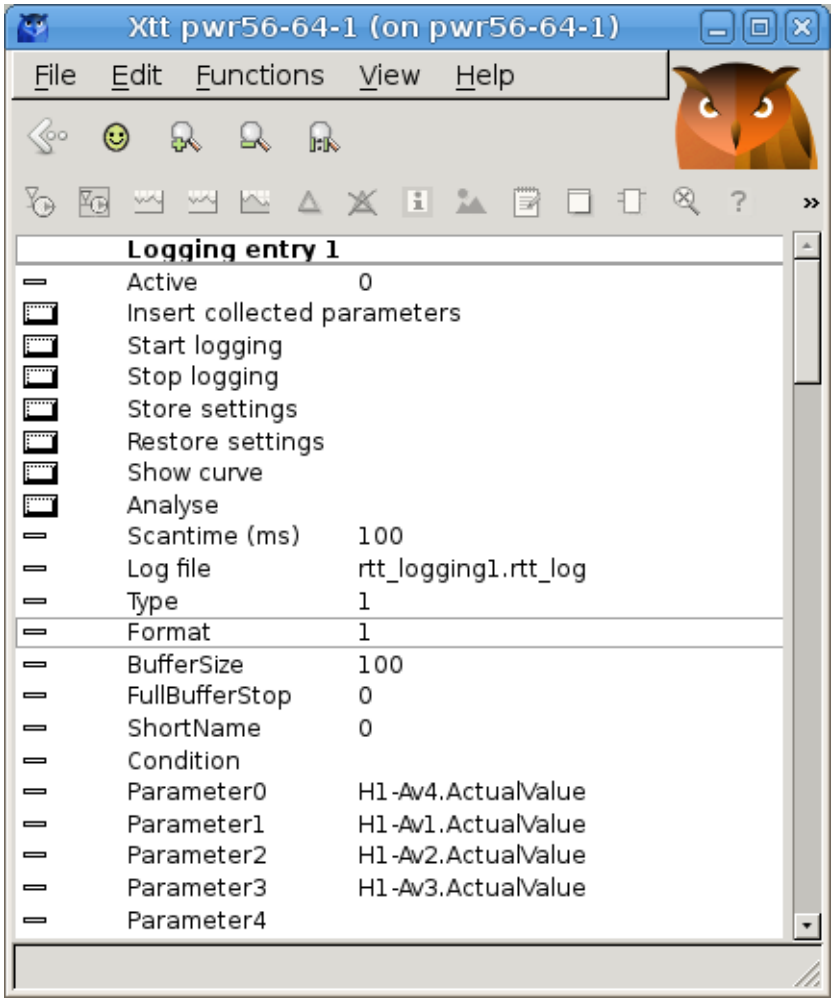
The new object Sim_ModelMLP implements a model with MLP (MultiLevel Perceptron). It will calculate an output value from a number of input values. To create the model, the output and the inputs are logged with the Xtt logging function, or stored on a sev server. The data is inserted in the sev_analyzer where the MLP is tranied. The model file is then copied to $pwr_exe and specified in the Sim_ModelMPL object.

Training example

Here is a simple example of how to train the MLP.

First we have to measure some data, and in this case it’s Av4 that our model should simulate, and

it’s dependent of Av1, Av2 and Av3. The xtt logging function is used for measuring, and the four attributes are collected an inserted in an logging entry. Format is set to 1 to get the correct time format in the log file, and the scan time is set to 100 ms as the MLP needs quite a lot of training data. The logging is started, and running for some time before it’s stopped.



Xtt logging

During the measurement it’s important to vary the input parameters (Av1, Av2 and Av3) and to get values in the whole range of these parameter and in all combinations.

The sev_analyzer is started from the ‘Analyze’ button with the measured data. The dataset only contains 1824 rows that is to small for the training, but we can use Edit/Multiply to increase the number of rows. The model parameter (Av4) should be positioned first in the dataset.

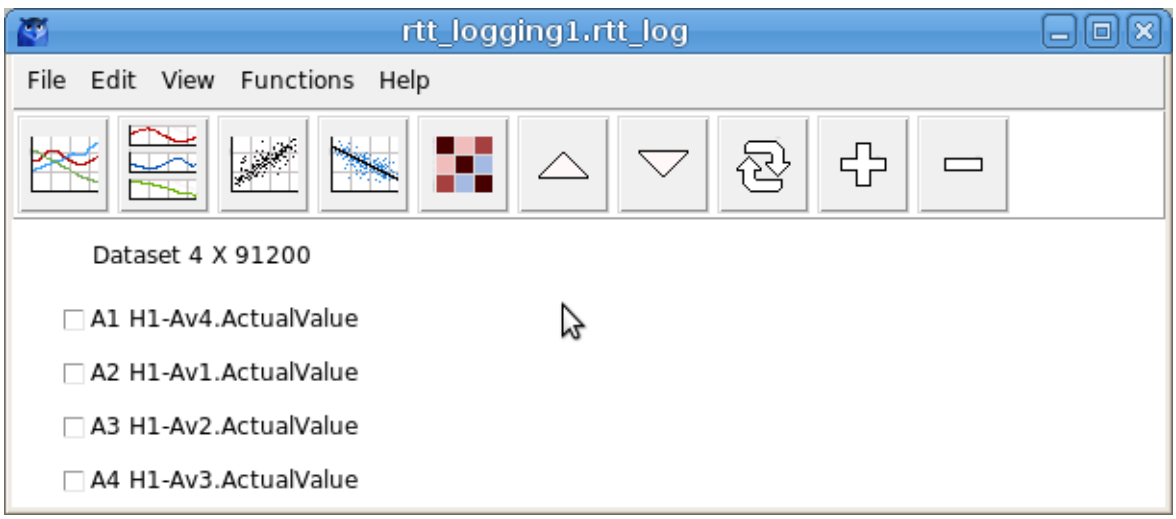


Fig analyzer

The MLP regressor is opened from *Functions/MLP Regressor* in the menu. We keep the default settings with three hidden layers of size 20, and activation function *tanh*, and start the training from *File/Create Model* in the menu. The result is a score of 0.986 which is pretty good (1 is perfect match). By changing the activation function to relu and adding another hidden layer, the score is increased to 0.994. In the result window below the model value is plotted with the sampled value.

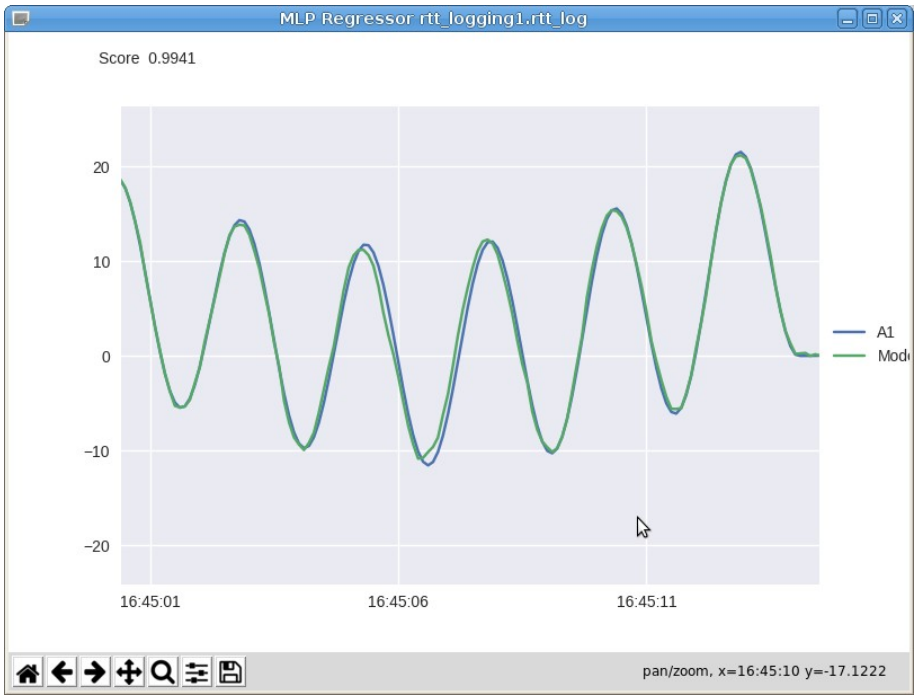


Fig Model result

The model is exported to file from File/Export Model in the menu. Save the file and copy it to $pwrp_exe.

The Sim_ModelMLP is divided in a main object and a function object. The main object is created in the configurator, and the ModeFile attribute is set to the name of the exported file.

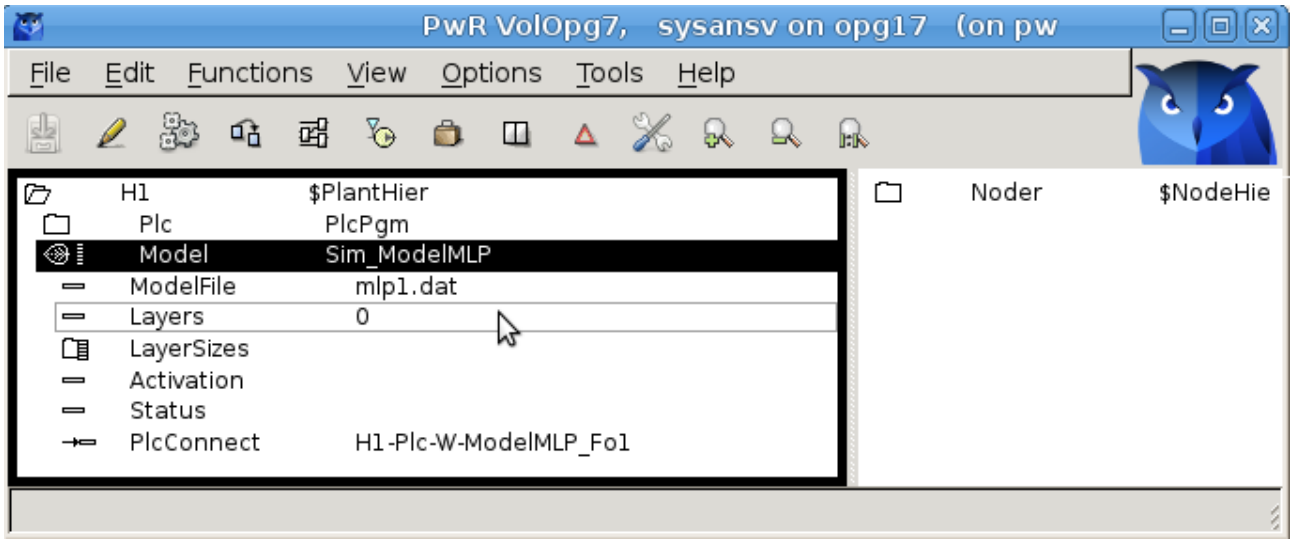


Fig Sim_ModelMLP object

The function object is inserted into a plc window and connected to the main object.

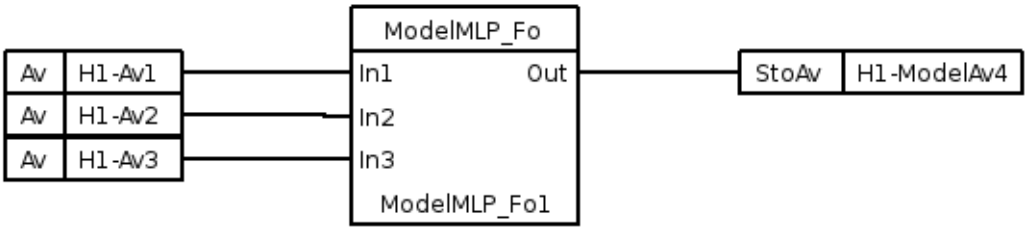


Fig Function object

MPC controller

An MPC (Model Predictive Control) controller contains a model of the process and uses this model to calculate the future response for different controller outputs. The output that will give the least error (difference between the set value and process value) is then selected and used as the next output for the controller.

In the example below, the level control from the demo project, the MPC is compared with a traditional PID controller.

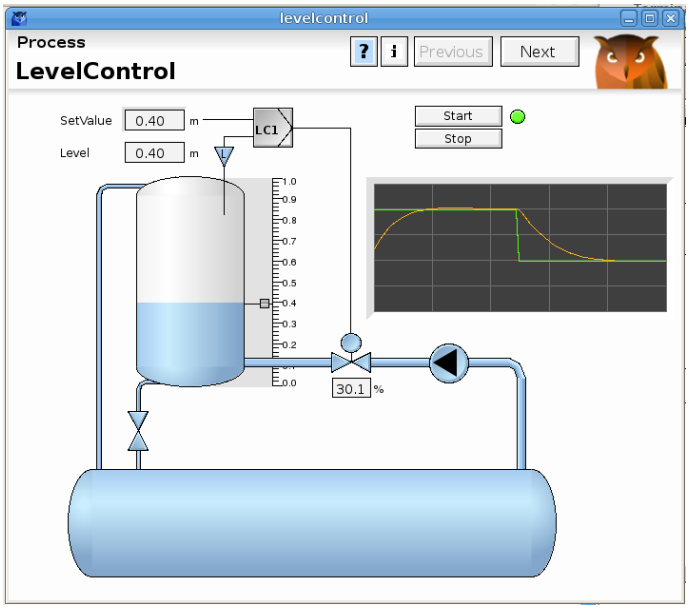


Fig Level control

A step response from a PID is displayed below. The output is mainly proportional to the error, and reaches a high value immediately after the step and then gradual decreases as the error decreases.

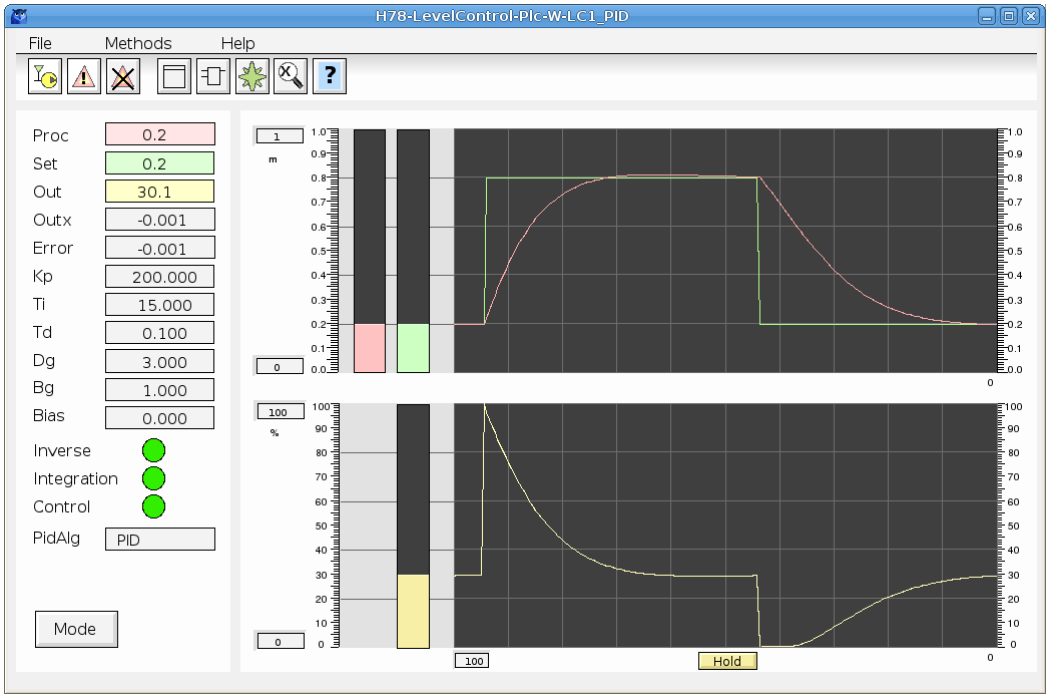


Fig Step response for PID

The MPC controller below has an entirely different approach. The output reaches the maximum value right after the step, and stays there until shortly before the process value reaches the set value.

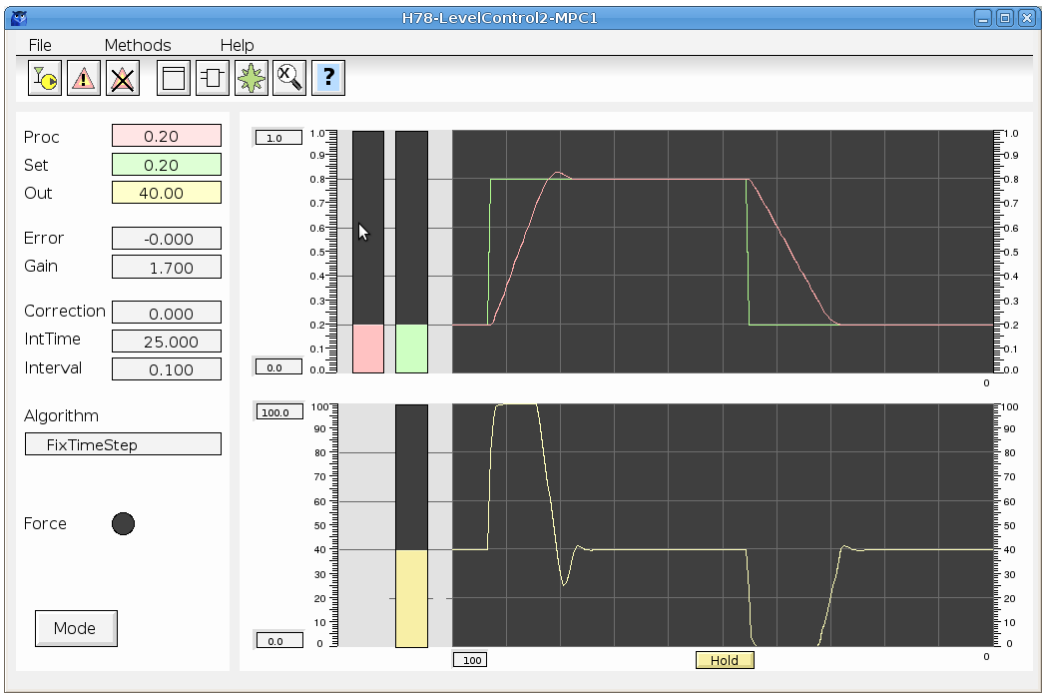


Fig Step response for MPC

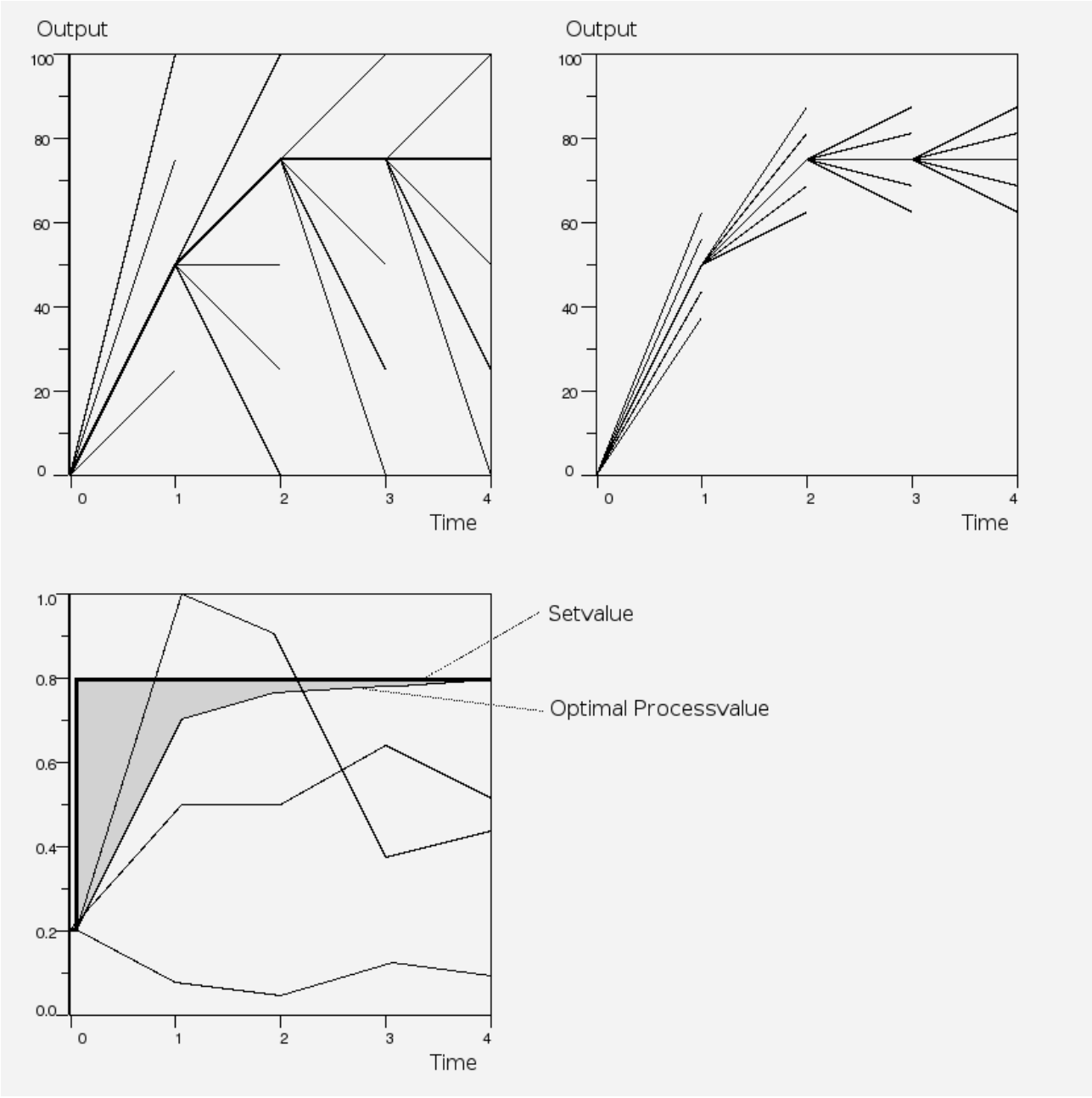
This is very close the the optimal behavior for a controller and shows the advantage of having the capability to foresee the future.

Prediction

The prediction is made iteratively by using the model to calculate the process value from the output value a number of time steps ahead. In the example below, 5 time steps is used, and in the initial step, 5 paths are calculated from the current output value. In the next and following time steps, each path is split in another five paths, although the split is only shown for one of the paths in the figure. This makes a total sum of $5^5 = 3125$ paths, and for each path the corresponding process value

path is calculated. In the lower graph the setvalue is drawn, which in this case is a step function, and a number of calculated process value paths. The most optimal path is regarded to be the one with the smallest area between the setvalue and processvalue (the gray area in the figure), and this path is chosen for the next iteration. In the next iteration (right graph below), the spread of the output paths is narrowed around the previously optimal path, and a new optimal path is calculated. Further iterations can give even more optimal paths.

In the end, it's actually only a small fraction of the first timestep that is used. The new output is set to the value at the next scantime of the controller, which is normally much shorter than a timestep. Then in the next scan, a new optimal path is calculated and the initial fraction of this path is used till the next scan.



Model

Two different types of models are implemented for the MPC, linear regression and MLP

Linear regression

A linear regression model can be created by the Multivariate analyzer from data logged in Xtt or fetched from a sev server. If the relation between the parameters are not linear, they first have to be

linearized by the conversion functions in the analyser. The controller is configured with a CompMPC object in the plant hierarchy, and a CompMPC_Fo object in the plc code. The CompModePID can be used as mode object, see figure below. The input parameters are connected to the A2 – A20 input pins. The model coefficients are inserted into AttrCoef0 and the AttrCoeff array, and also the linearization has to be specified in BaseAttrRelation and TightAttrRelation.

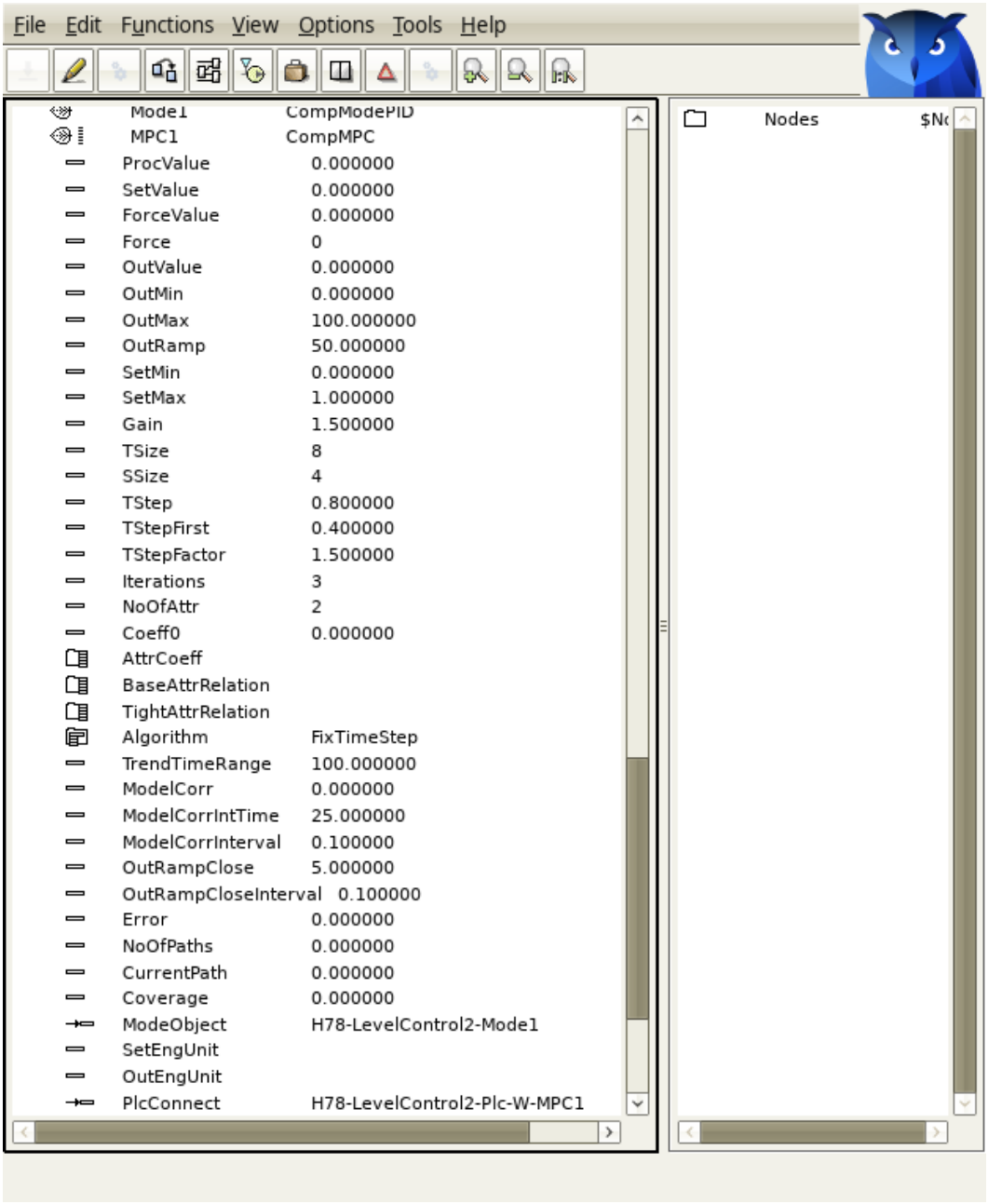


Fig CompMPC object

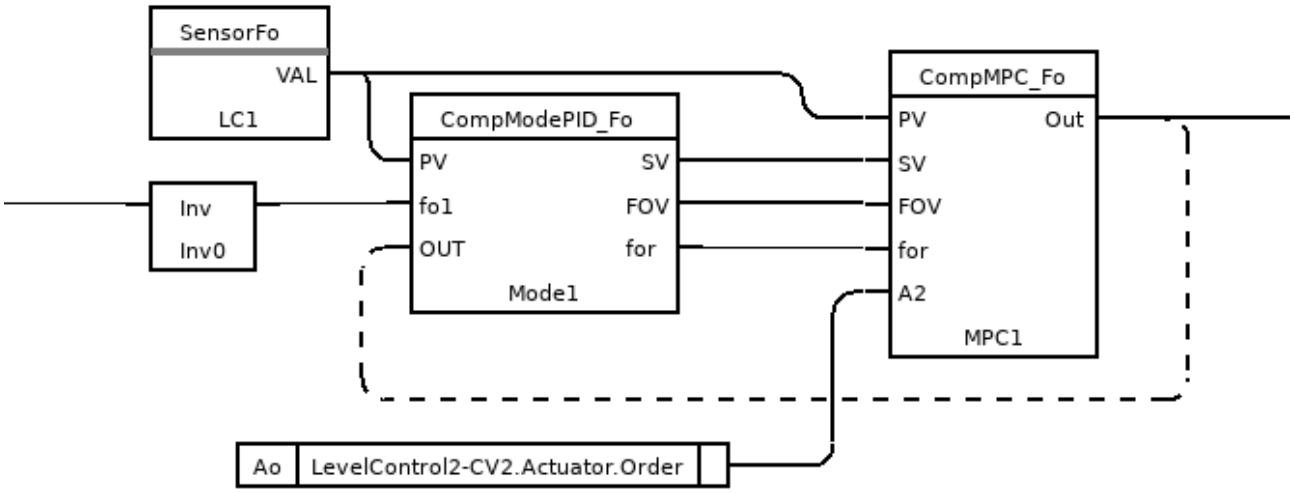


Fig Function object for CompMPC

An integration term is added to the model value to eliminate any residual error. This is configured with ModelCorrIntTime. It's also possible to define a window where the integration is active, ModelCorrInterval. The integration term will only be used when the error is less than this value.

MLP

CompMPC_MLP and CompMPC_MLP_Fo contains a neural network in shape of an MLP. The MLP is trained by the Multivariate analyzer, see the Sim_ModelMLP object above.

The MLP can handle nonlinear relations, thus no conversion of the input parameters are required. One problem though is that the MLP can't handle accumulative processes as there is no time relation involved in the MLP training. A solution to this is implemented where inputs that have integral relation are trained also with time shifted values. The time shifts should equal the time steps in the prediction.

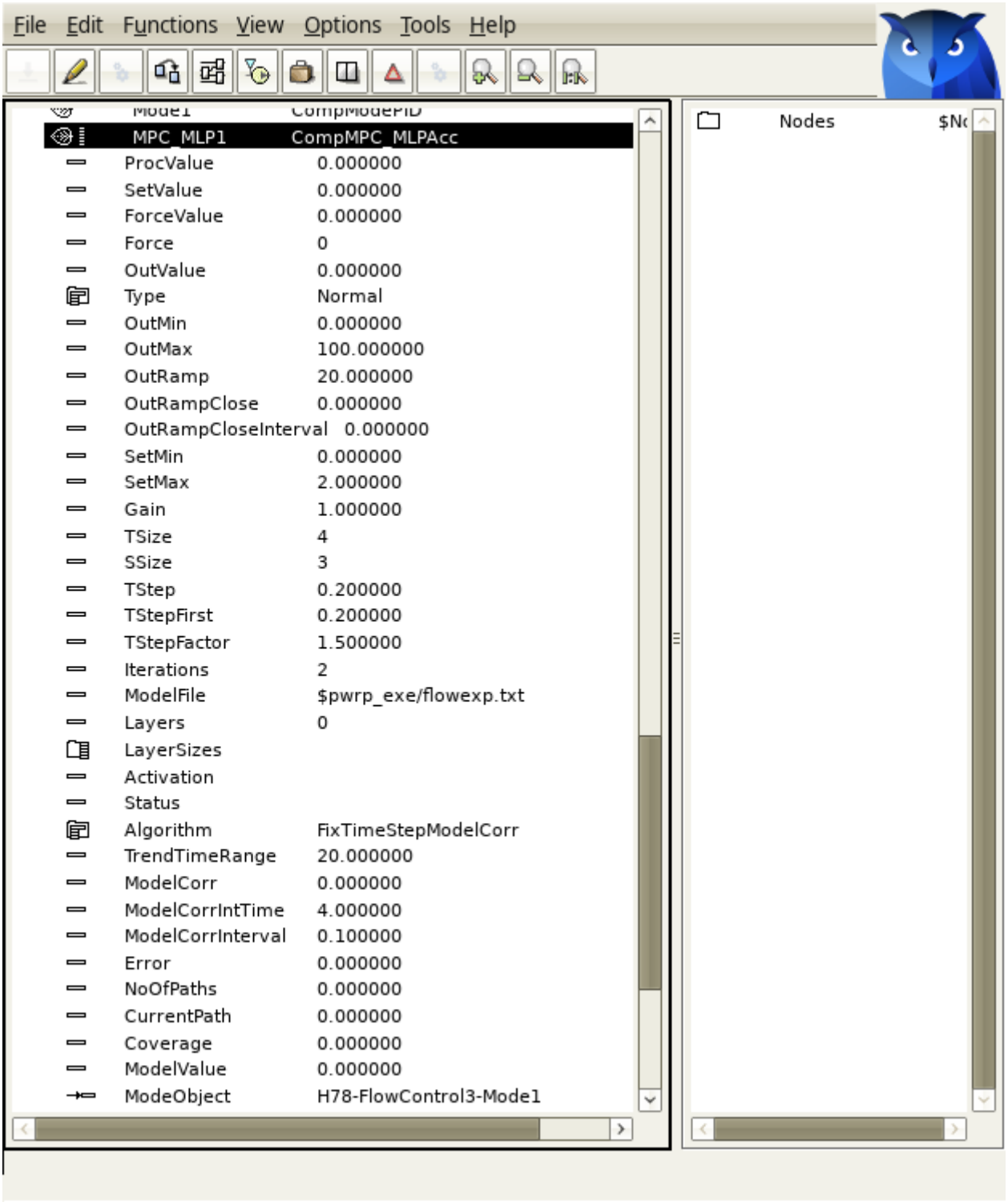


Fig CompMPC_MLP object

Predictive maintenance supervision

Beta version.

Predictive maintenance is maintenance that monitors the performance and condition of equipment during normal operation to reduce the likelihood of failures. One way to do this is to supervise the relation between correlated process parameters and check that the relation doesn’t change over time.

The relation is calculated with linear regression by the multivariate analyzer. If the parameters don’t have linear dependencies they first have to be linearized. The linear regression will give a set of

coefficients, and by cyclically fetching the parameters from the history database, applying the linearization formla on the parameters, and do the linear regression, the trend of the coefficients can be monitored. If they exceed limit values, alarm messages are be sent to operators or maintainers.

In the level control example from the demo project (see fig in MPC chapter above) the level is dependent on the in and out flow. Before making a linear regression, the in and out flow has to be linearized by a time integration. This is done with the conversion tool in the analyzer. The formula for the conversion is stored on file and the linear regression is executed.

The same sequence with linearization and linear regression will now be performed cyclically by the maintenance supervision server, rt_maimtsupserver.py. A supervision is configured with a LinRegSup object where the parameters and the formula file is specified. The server will fetch fresh values for the parameters from the history database, then execute the linearization and linear regression. The calculated coefficients will be compared with limit values in the Limit array of the LinRegSup object. The Limit array also contains DsupComp object to configure alarm messages that will be sent if any coefficients is out of range.

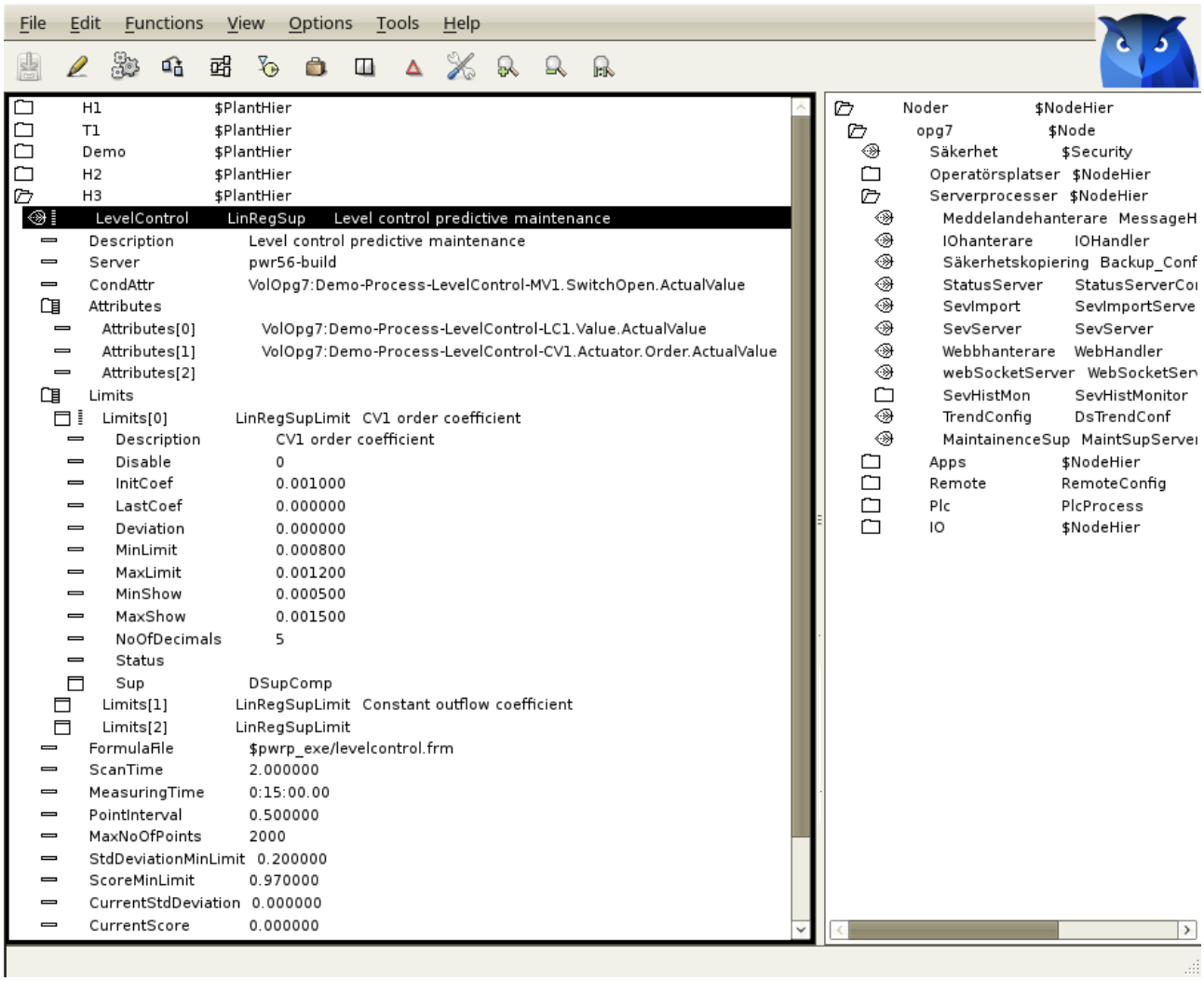


Fig Configuration of supervision with a LinRegSup object

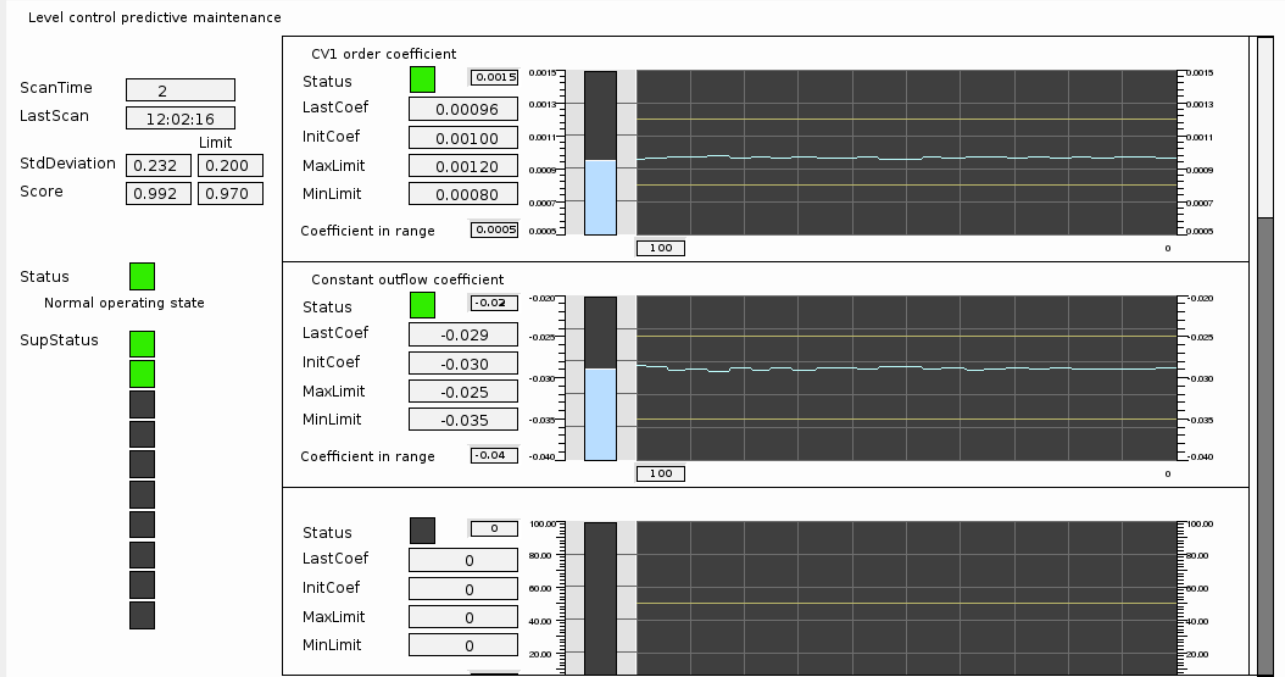


Fig Object graph for LinRegSup showing the trends for the coefficients

To ensure the calculated coefficients are valid, some precautions can be made.

- There is a condition attribute CondAttr that can be specified to avoid calculations when the process is stopped. History data will only be used when the condition attribute is true.
- If there are no variations in the process during the time interval for history data, there will be a large uncertainty in the calculated coefficients. Therefore it’s possible to set a min value for the standard deviation of the y value. If the standard deviation is to low, no linear regression is performed.
- It is also possible to set a min value for the score of the linear regression. If the score is to low, no comparision of the coefficients are made.

The supervision server process, , rt_maimtsupserver.py, is configured wih a MaintSupServer object in the node hierarchy.

Random function object

Function object that returns a random value in a specified interval.

LightDv, LightAv, LightIv, LightSv

Value object for digital, analog, integer and string values respectively. ActualValue in these objects are not I/O copied. The values can be fetched in the plc code with the GetXp where X is D, A, I or S. Setting is made with StoXp or similar objects, and a setting will be detected immediately by GetXp objects and not in the next scan as for I/O copied objects. Thus the execution order is of great importance when handling these objects.

New Types and Classes

$MountDynObject

Mount object for dynamic objects.

$Block

Root object for a block object.

$SubBlock

Defines an attribute object in a block object.

$BlockAttribute

Base object for Defines attribute definition objects in block objects.

BlockAttrBoolean, BlockAttrInt32, BlockAttrFloat32, BlockAttrString

Defines an attribute in a block object of type Boolean, Int32, Float32 and String80 respectively.

MaintSupServer

Configuration object for the predictive maintenance supervision server.

LinRegSup

Supervision of parameter relations with linear regression for predictive maintenance.

CompMPC, CompMPC_Fo

MPC controller with linear regression model.

CompMPC_MLP, CompMPC_MLP_Fo

MPC controller with MLP model.

SevImportServer

Configuration of the import server on a sev_server.

Random

Function object that returns a random value.

LightDv, LightAv, LightIv, LightSv

Value objects for digital, analog, integer and string values that are not I/O copied.

Sev:

New class volume for sev module.

SevExpItem

Base class for dynamic objects to display value and info of exported variables on a sev server.

SevItem

Base class for dynamic objects to display value and info of history variables on a sev server.

SevExpItemBoolean, SevExpItemFloat32 etc

Dynamic objects to display value and info of exported variables on a sev server.

SevExpItemBooleanArray, SevExpItemFloat32Array etc

Dynamic objects to display value and info of exported array variables on a sev server.

SevItemBoolean, SevItemFloat32 etc

Dynamic objects to display value and info of history variables on a sev server.

Sim_ModelMLP, Sim_ModeMLP_Fo

MLP model object.

Sim_SignalGenerator, Sim_SignalGeneratorFo

Simulation object to create signals with different wave form, filter and noise.

Sim_CylinderTank, Sim_CylnderTankFo

Simulation of the level in a cylinder tank.

Sim_Furnace, Sim_FurnaceFo

Simulation of a the temperature in a furnace.

Modified Classes

PlotGroup

Attributes ButtonText, TimeRange and Layout added.

Upgrade procedure

The upgrading has to be done from any V5.6. 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

The upgrade procedure is to dump the database with reload.sh, change the version of the project in the projectlist, and then execute the script upgrade.sh.

NOTE !!

Do not activate Update Classes.

If the previous version should be kept, first make a copy of the project.

Make a copy of the project

Do `sdf` to the project and start the administrator

```
> pwra
```

Now the Projectlist is opened. Enter edit mode, login as administrator if you lack access. Find the current project and select Copy Project from the popup menu of the ProjectReg object. Open the copy and assign a suitable project name and path. Save and close the administrator.

Dump the databases

Execute the first pass, *dumpdb*, in the script *`reload.sh`*.

```
> reload.sh

  reload.sh   Dump and reload of database.

  Arguments   Database or databases to reload.
              I no arguments is supplied, all databases will be
              reloaded.

  Pass

    dumpdb       Dump database to textfile $pwrp_db/'volume'.wb_dmp
    classvolumes Create structfiles and loadfiles for classvolumes
    renamedb     Rename the old database
    loaddb       Load the dump into the new database
    compile      Compile all plcprograms in the database
    createload   Create new loadfiles.
    createboot   Create bootfiles for all nodes in the project.


-- Reloading volume  directory volopg2

Pass: dumpdb classvolumes renamedbloaddb compile createload createboot

Enter start pass [dumpdb] >

------------------------------------------------------------------------
 Pass dump database
------------------------------------------------------------------------
Do you want to continue ? [y/n/go] y
ls: cannot access /data0/pwrp/opg2/common/db/*.wb_dmp: No such file or
directory
Dumping volume directory in /data0/pwrp/opg2/common/db/directory.wb_dmp
...
I Database opened /data0/pwrp/opg2/common/db/volopg2.db
ls: cannot access /data0/pwrp/opg2/common/db/*.wb_load: No such file or
directory



------------------------------------------------------------------------
 Pass create structfiles and loadfiles for classvolumes
------------------------------------------------------------------------
Do you want to continue ? [y/n/go] n
setdb is obsolete
>
```

Check that the one dumpfile is created for every rootvolume

```
> cd $pwrp_db
> ls -l *.wb_dmp
-rw-rw-r-- 1 cs pwrp 7467 2010-03-26 16:32 volopg2.wb_dmp
```

Linux release upgrade

If you are using an older Ubuntu version upgrade the linux release and install the pwr55 package.

Change version

Enter the administrator and change the version of the project to V5.5.0. Save and close the administrator.

upgrade.sh

Do `sdf` to the project.

upgrade.sh is a script that is divided into a number of passes. After each pass you you have to answere whether to continue with the next pass or not.
Start the script with

```
> upgrade.sh
```

Start from the savedirectory pass.

```
Enter start pass [savedirectory] >
```

savedirectory

Save the directory volume.

classvolumes

Create loadfiles and structfiles for the class volumes.

renamedb

Store the old databases under the name `$pwrp_db/'volumename'.db.1.`

loaddb

Create databases and load the dumpfiles into them.

compile

Compile all the plc programs.

createload

Create loadfiles for the root volumes.

createboot

Create bootfiles for all nodes in the project.

If the project contains any application programs, these has to be built manually.

Delete files from the upgrading procedure:

`$pwrp_db/*.wb_dmp.*`
`$pwrp_db/*.db.1` (old databases, directories which content also should be removed)

List example

```
>
> sdf opg2
Setting base /data0/x5-6-0/rls
>
> upgrade.sh

  upgrade.sh  Upgrade from V5.6 to V5.7


  Pass

    savedirectory  Save directory volume.
    classvolumes   Create loadfiles for classvolumes.
    renamedb       Rename old databases.
    loaddb         Load dumpfiles.
    compile        Compile all plcprograms in the database
    createload     Create new loadfiles.
    createboot     Create bootfiles for all nodes in the project.
    createpackage  Create distribution packages for all nodes in the
                   project.

-- Upgrade opg2

Enter start pass [savedirectory] >

------------------------------------------------------------------------
 Pass save directory volume
------------------------------------------------------------------------
Do you want to continue ? [y/n/go] y
-- Processing line: 270
-- Building volume directory
I Volume directory loaded
I Database opened /data0/pwrp/pwrtestloc/src/db/directory.wb_load
-- Saving file /data0/pwrp/pwrtestloc/src/db/directory.wb_load ->
/data0/pwrp/pwrtestloc/src/db/directory.wb_load.1
%WNAV-E-MSG, Session saved

------------------------------------------------------------------------
Pass create structfiles and loadfiles for classvolumes
------------------------------------------------------------------------
Do you want to continue ? [y/n/go] y
ls: cannot access /data0/pwrp/opg2/src/db/*.wb_load: No such file or
directory
------------------------------------------------------------------------
Pass convert dumpfiles
------------------------------------------------------------------------
Do you want to continue ? [y/n/go] y
```

```
/data0/pwrp/opg2/src/db/volopg2.wb_dmp

------------------------------------------------------------------------
Pass rename old databases
------------------------------------------------------------------------
Do you want to continue ? [y/n/go] y
-- Saving file /data0/pwrp/opg2/src/db/volopg.db ->
/data0/pwrp/opg2/src/db/volopg.db.1

------------------------------------------------------------------------
Pass load database
------------------------------------------------------------------------
Do you want to continue ? [y/n/go] y
-- Loading volume volopg
...
-- Processing line: 57
-- Building volume directory
I Volume directory loaded
I Database opened /data0/pwrp/opg2/src/db/directory.wb_load
-- Processing line: 200
-- Building volume VolOpg
I Volume VolOpg loaded
Berkeley DB 4.6.21: (September 27, 2007)
info put: 0
Berkeley DB 4.6.21: (September 27, 2007)
info get: 0
int rc =  m_txn->abort(): 0

------------------------------------------------------------------------
 Pass convert objects in loaded database
------------------------------------------------------------------------
Do you want to continue ? [y/n/go] y
-- Processing line: 200
-- Building volume directory
I Volume directory loaded
I Database opened /data0/pwrp/opg2/src/db/directory.wb_load
-- Saving file /data0/pwrp/opg2/src/db/directory.wb_load -> /data0/pwrp/
opg2/src/db/directory.wb_load.1
Print wbl
-- Writing line: 200
E Error, Volume 'VolOpgSev' is not yet created.
%WNAV-E-MSG, Session saved

------------------------------------------------------------------------
Pass compile plcprograms
------------------------------------------------------------------------
Do you want to continue ? [y/n/go] y
...
Berkeley DB 4.6.21: (September 27, 2007)
info get: 0
I Database opened /data0/pwrp/opg2/src/db/volopg.db
-- Plc window generated            F1-Z1-Plc-W
-- Plc window compiled for x86_linux optimized -O3 F1-Z1-Plc-W
-- Plc plcpgm compiled for x86_linux optimized -O3 F1-Z1-Plc
-- Plc window generated            F1-Z2-Plc-W
-- Plc window compiled for x86_linux optimized -O3 F1-Z2-Plc-W
```

```
-- Plc plcpgm compiled for x86_linux optimized -O3 F1-Z2-Plc

------------------------------------------------------------------------
Pass create loadfiles
------------------------------------------------------------------------
Do you want to continue ? [y/n/go] y
-- Removing old loadfiles
rm: cannot remove `/data0/pwrp/opg2/bld/common/load/ld_vol*.dat': No
such file or directory
...
Berkeley DB 4.6.21: (September 27, 2007)
info get: 0
I Database opened /data0/pwrp/opg2/src/db/volopg.db
-- Building archive for volume: 000_001_001_012
-- Archive built for volume: 000_001_001_012

-- Working with load file volume 'VolOpg'...
-- Open file...
-- Successfully created load file for volume 'VolOpg'
-- 26 objects with a total body size of 21976 bytes were written to new
file.

Before this pass you should compile the modules included by ra_plc_user.


------------------------------------------------------------------------
Pass create bootfiles
------------------------------------------------------------------------
Do you want to continue ? [y/n/go] y
-- Creating bootfiles for all nodes

ProviewR is free software; covered by the GNU General Public License.
You can redistribute it and/or modify it under the terms of this
license.

ProviewR is distributed in the hope that it will be useful
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

-- Creating bootfile for node opg
      plc_opg_0507_00011
-- Plc thread generated priority 0, scantime   0.10000 s, 2 plcpgm's
-- Plc process compiled for x86_linux optimized -O3 Dummy
-- Plc program linked for x86_linux  node plc_opg_0507
-- Creating bootfile for node aristotle
      plc_aristotle_0517_00011
-- Plc thread generated priority 0, scantime   0.10000 s, 2 plcpgm's
-- Plc process compiled for x86_linux optimized -O3 Dummy
-- Plc program linked for x86_linux  node plc_aristotle_0517

-- The upgrade procedure is now accomplished.

setdb is obsolete
>
>
```