ProviewR

OPEN SOURCE PROCESS CONTROL

Guide to Storage Environment

Version 6.1.5

2 Introduction

A task of increasing importance for the automation systems is to store process data. The possibility to store large amounts of data has given rise to new functionality in terms of development and optimization of processes, predictive maintenance, calculation of models for simulation etc.

This document describes storage of process history in ProviewR and how this is configured.

3 Process data

Storing of process history data means that signals and other data is stored cyclically in a database, and from there can be fetched to displayed in curves or analyzed in other contexts, eg for predictive maintenance or process development.

The storage is configured with the object SevHist and SevHistObject. SevHist configures storage of one attribute, and SevHistObject of a whole object. In the SevHist object you state how often the value should be stored, and for how long.

The storage is handled by two processes, a client process, rt_sevhistmon, that collects data and sends it to a server process, sev_server, that stores the data in the database. The client process can send data to several different server processes, and the server process can receive data from several different client processes, The client and server process can run in the same node, or in different nodes. For test and troubleshooting you can start a server process on a process station that stores data from its own node. For larger amounts of data with storage over several years, you appropriately create a dedicated storage station, that stores data from several nodes.

3.1 Client

rt_sevhistmon is the client process that collects the process values in one node and sends it to the server. It's configured with a SevHistMonitor object in the node hierarchy. Below the SevHistMonitor object you place a SevHistThread object, that contains cycle time and server node. Each SevHist and SevHistObject object is connected to a thread object and thus will stored on the node, and with the cycle time specified in the thread object.

The thread object also contains a ServerThread attribute, by which the storage can be directed to a specific thread in the server process. This can be used to spread the load over different threads in the server process.

3.2 Server

The server process, sev_server, receives data from the client processes an stores it in the database. If the database doesn't exist, it will be created with the required tables. Then a handshake with the client processes is performed, where client processes sends information about all the attributes and objects that should be stored. The cyclic transfer of process data from the client process to the server process then starts.

The server process also can answer requests of history date for an attribute in a specific time interval.

Data can be stored in different types of databases, MariaDB (MySQl), Sqlite and HDF5.

MariaDB/MySQL

MariaDB is a clone of MySql that was created when MySQL was taken over by Oracle. They

should be compatible and exchangeable with each other. In recent Linux releases often MariaDB is installed. Many configuration alternatives in ProviewR is still named MySQL but should also be used for MariaDB.

MariaDB should be used for larger databases and for databases for permanent storage.

Sqlite

Sqlite is a small, fast database that doesn't require installation of a server. It can be used for smaller databases for test and troubleshooting.

HDF5

HDF5 is a file format to store large amounts of data. The absence of journaling doesn't make it suitable for permanent storage.

Alarms and events

Storage of alarms and events can be achieved with the SevHistEvent object. The object contains an EventSelectList where hierarchies for alarms that should be stored are specified.

4 Configuration

The storage process history for a single attribute is configured with a SevHist object, and storage for a whole object with a SevHistObject object. Furthermore the client process is configured with SevHistMonitor and SevHistThread objects, and the server process with a SevServer object.

4.1 SevHist

Attributes that should be stored in the history database is configured with the SevHist object. The attribute to store is specified in Attribute in the SevHist object. The SevHist object is normally positioned below the object that is stored. If the object contains an ActualValue attribute. this will automatically be inserted into the SevHist object.

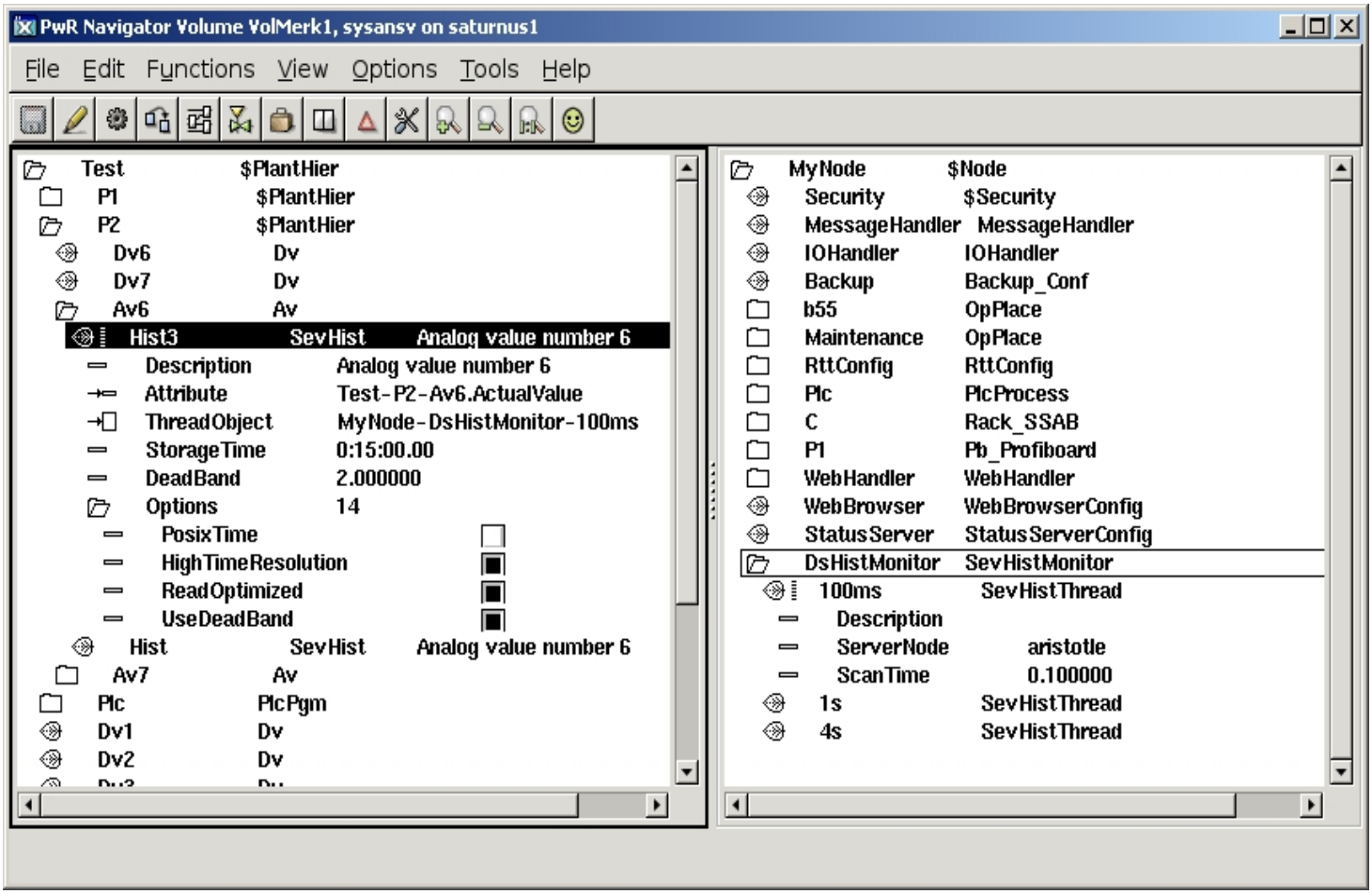


Fig Configuration with SevHist, SevHistMonitor och SevHistThread objekt

4.2 SevHistObject

SevHistObject stores all the attributes in an object into a single table.

It is recommended to create a specific class for this purpose, as existing classes often contains attributes that shouldn't be stored.

4.3 Server threads

By configuring threading of the server process, the performance can be substantially increased as the load is spread on several different threads.

Threading is implemented for MariaDB/MySSQL.

The threads are configured in the client by stating a thread number in SevHistThread object. A thread can be numbered with an arbitrary positive number, and all SevHistThread objects with the same thread number will be handled by the same server thread.

In the object graph of the SevServer object, the load of each thread is displayed, and by altering the thread number in the SevHistThread objects, one can make sure that no server thread is overloaded.

Server threads also have to be configured in the server process by setting the attribute UseServerThreads in the SevServer object to 1.

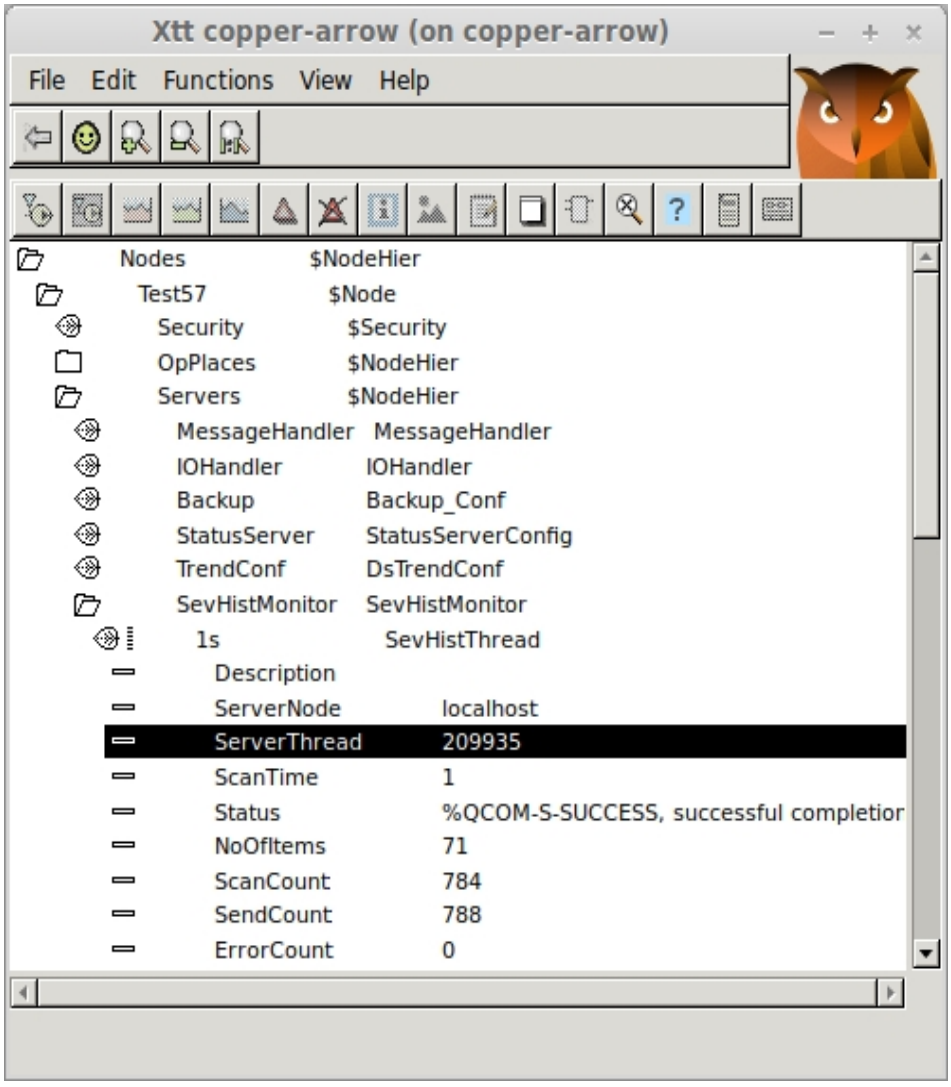


Fig Configuration of thread number in the SevHistThread object.

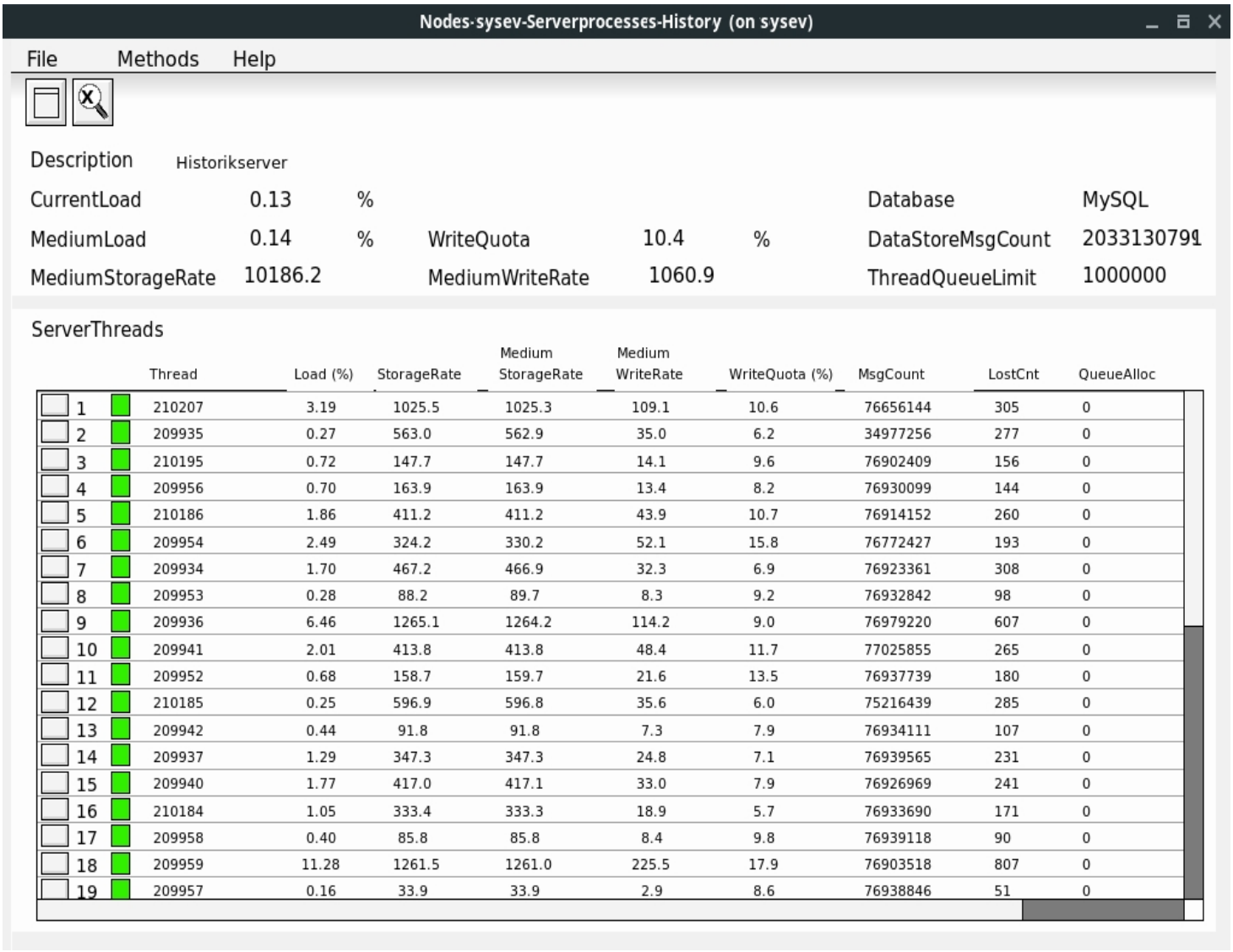


Fig Object graph for SevServer object displaying the server threads.

4.4 Deadband

Deadband can be configured on analog, digital and integer signals, and means the a certain change of the value is required before a new value is stored into the database. By setting a deadband the disk space used to store a signal can be substantially reduced.

The deadband is configured by setting Deadband and ReadOptimized in Options in the SevHist object, and state the size of the deadband in the Deadband attribute.

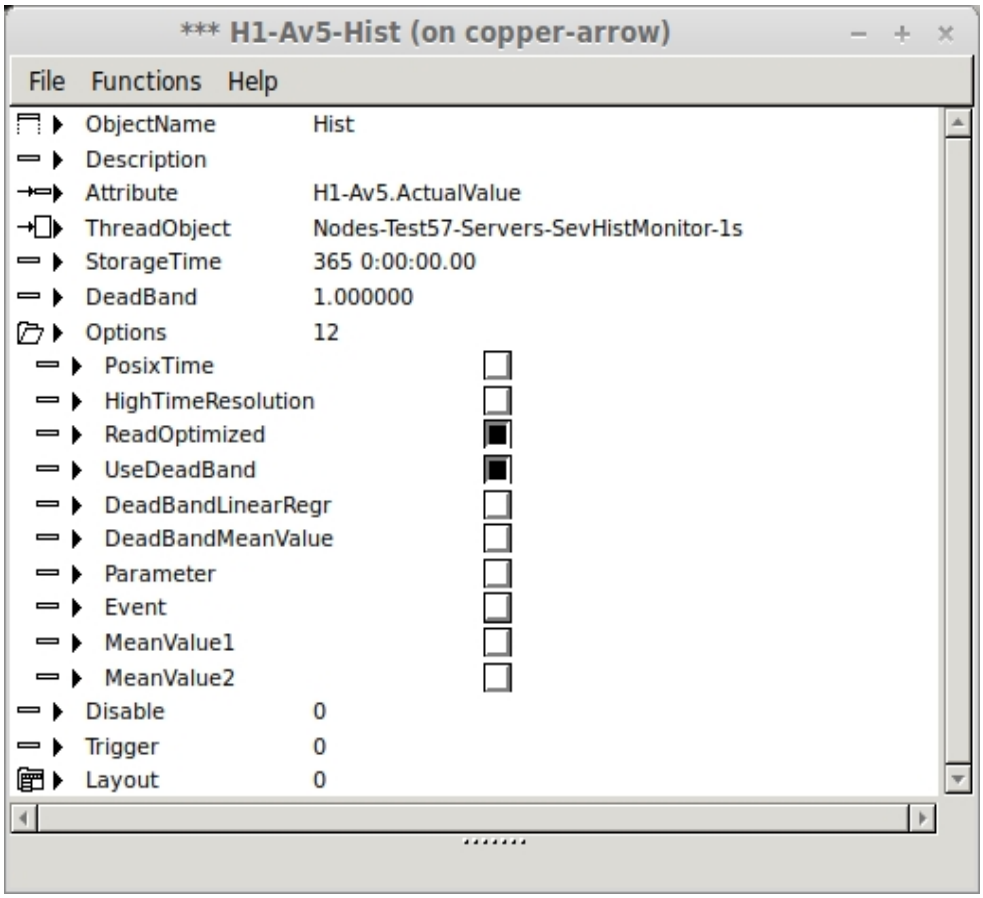


Fig Deadband configuration.

For deadband on digital signals, set Deadband to 0.5.

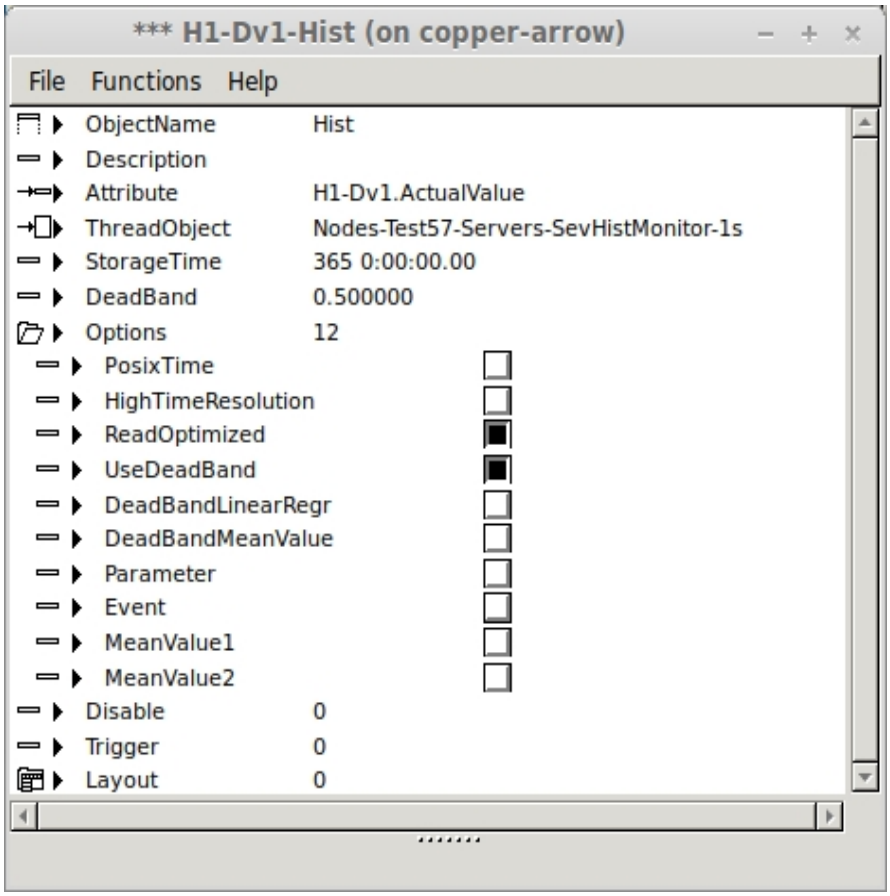


Fig Deadband configuration for a digital signal

4.5 Deadband with linear regression

This is a two dimensional deadband that also works on ramps.
With linear regression a straight line is calculated from the latest store value, and
as long as no value deviates more than the deadband from the line, no new value is stored.
This will even more reduce the required disk space.

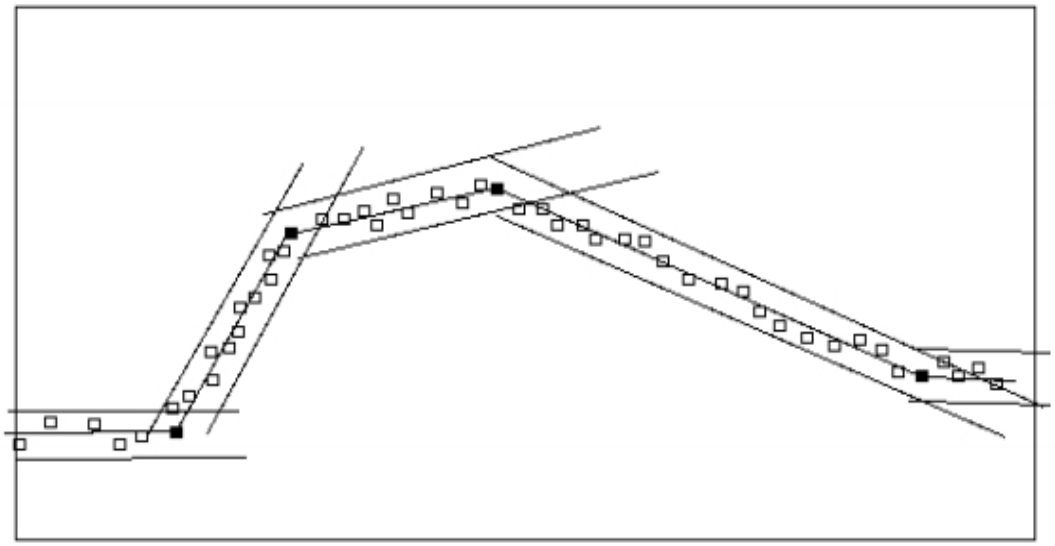


Fig Deadband with linear regression

Deadband with linear regression is configured by setting Deadband, DeadbandLinearRegr and ReadOptimized in Options in the SevHist object, and supplying the size of the deadband in Deadband.

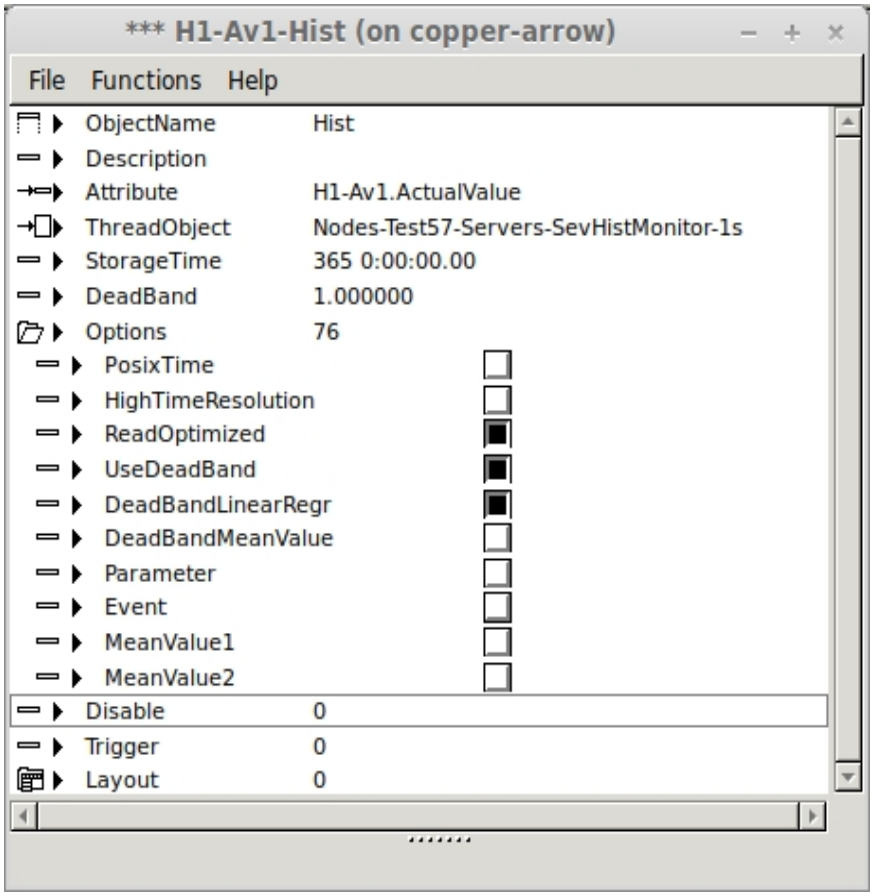


Fig Configuration of deadband with linear regression

4.6 Event triggered storage

By setting Event in Options in the SevHist object, it is possible to control when the storage is performed. When Trigger in the SevHist object is set to 1, the current value is sent to the server process. Trigger is reset when the value is sent.

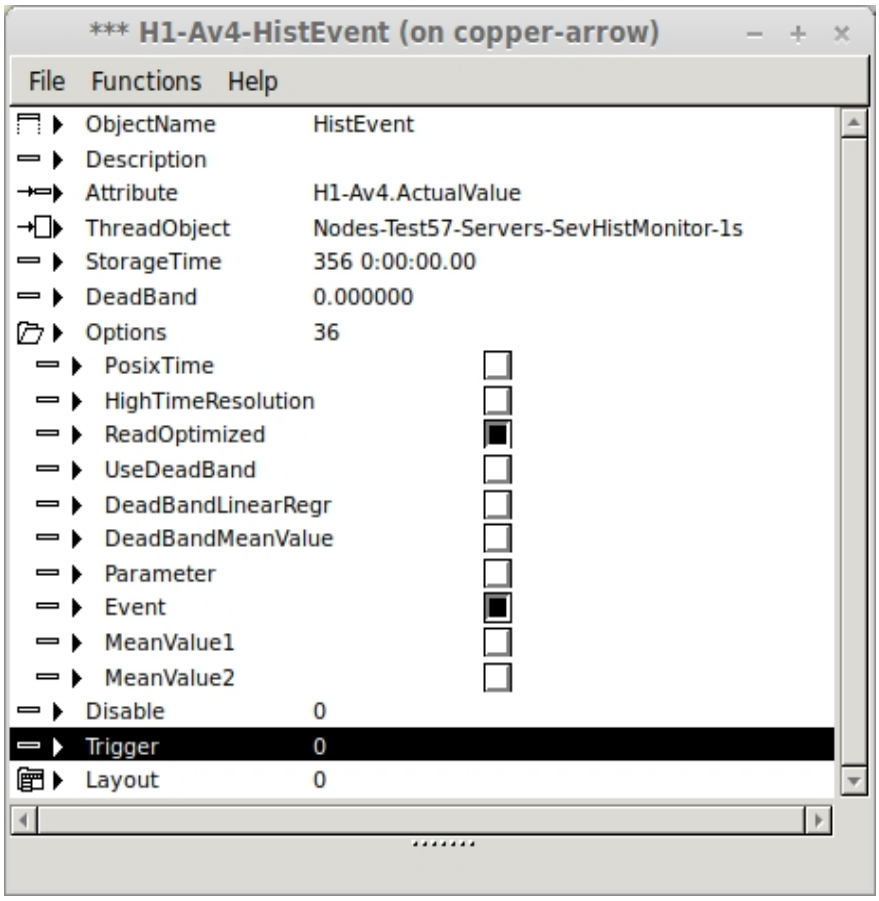


Fig Configuration of event triggered storage

4.7 Meanvalue calculation of stored signals

The server process can perform a meanvalue calculation of a stored signal, and this is configured by activating MeanValue1 or MeanValue2 in Options in the SevHist object.

The server can make a calculation with two different times, and these are set in MeanValueInterval1 and MeanValueInterval2 in the SevServer object. In SevHist.Options you select which of these times the calculation should be executed with.

The meanvalue is stored in the item tree, and from there is can be referred from Ge graphs and applications with the suffix '.__MeanValue', eg 'pwrNode-sev-H1-Av1.ActualValue.__MeanValue'.

4.8 Item tree

Each stored signal is represented of an item. All items are displayed in an item tree that is placed under pwrNode-sev in the realtime database. In the items tree, the signals are ordered in their original hierarchy, and the last received value is displayed. More information about the items is displayed by clicking with Shift+Click on the value, or with Shift+Arrow left on the keyboard.

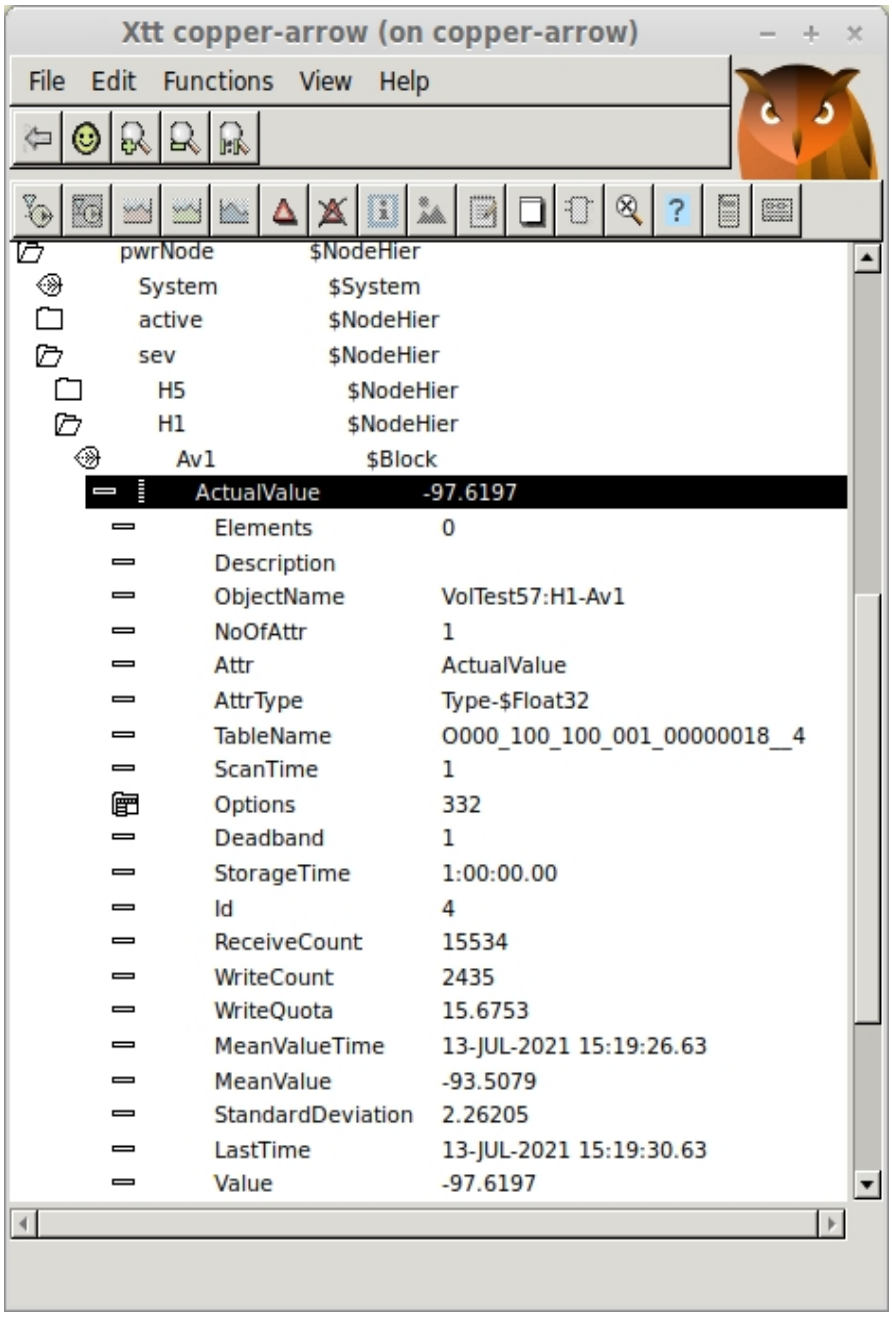


Fig An item in the item tree

4.9 Mounting of the item tree

The item tree is built of objects of type $Block and $BlockAttribute to recreate the original hierarchy and object structure to some extent. By mounting the hierarchies under pwrNode-sev on the top level you can also recreate the original object and attribute names and use them for references in graphs and applications.

The mount is made with mount objects of type $MountDynObjects as the objects in the item tree are dynamic objects.

When the mount is done, signals can be referenced with their original names. This makes it easier to refer to the signals in graphs and applications.

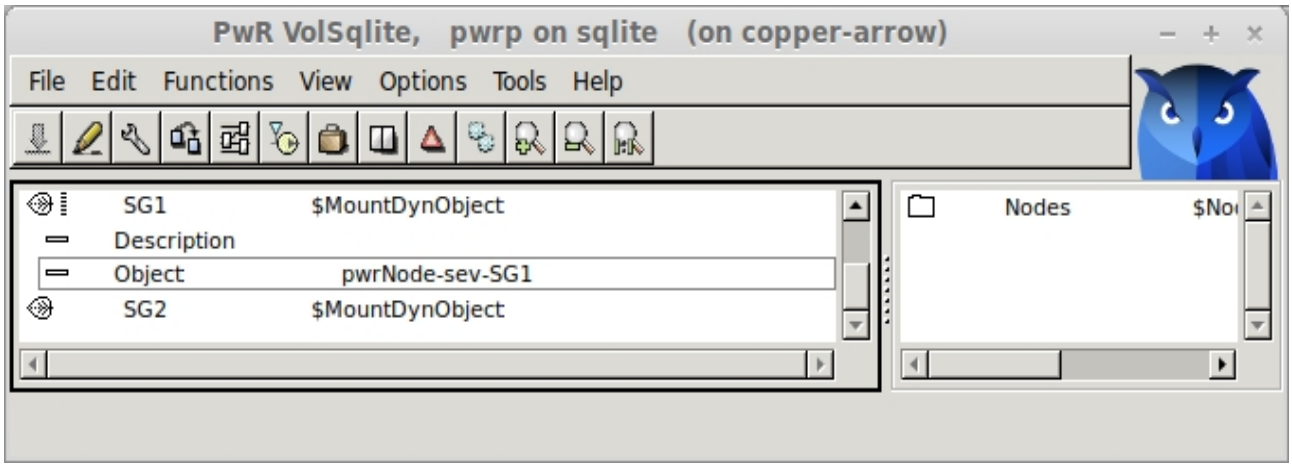


Fig Mount of hierarchy in the item tree

4.10 Refer to data in the item tree

Data in the item tree can be referred to with the suffix '.__'dataname''. For example the mean value for H1-Av1 is referred to by

```
pwrNode-sev-H1-Av1.ActualValue.__MeanValue
```

If there is a mount of H1, pwrNode-sev is superfluous and the reference can be made with

```
H1-Av1.ActualValue.__MeanValue
```

It's possible to fetch the value and subscribe to it from c++ and Python code. In Ge graphs the value type should be added

```
H1-Av1.ActualValue.__MeanValue##Float32
```

Here are some examples of other values that can be referenced

H1-Av1.ActualValue	Last received value.
H1-Av1.ActualValue.__TableName	Name to table where the value is stored.
H1-Av1.ActualValue.__StandardDeviation	Standard deviation for the mean value.
H1-Av1.ActualValue.__LastTime	Time for last received value.

4.11 Plc programming

Item data can also be fetched in the plc program. As the data reside in dynamic objects the GetExt objects has to be used, eg GetExtFloat32, GetExtBoolean etc.

In the example below, the momentary value for H1-Av1.ActualValue and the mean value for H1-Av2.ActualValue are added and put into H3-Av3 that is a local object in the server node.

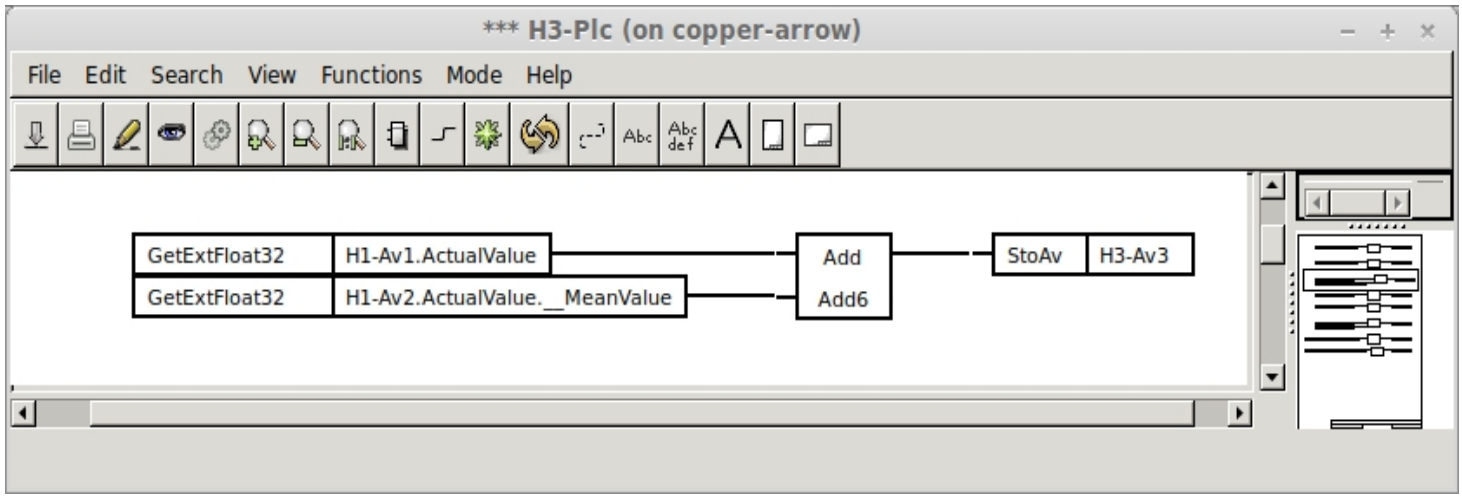


Fig Plc code with item data

4.12 Sev export

Sev export makes it possible to export data to the sev server that doesn't need to be stored in the history database, but nevertheless should be available to be displayed in graphs and reports.

The export is configured with SevExport objects. A SevExport object exports one attribute. As for the SevHists objects, it is the rt_sevhistmon process that collects the attributes and sends them to the server node. In this case the receiver process is sev_import, that is configured with a SevImportServer object in the sev server node. sev_import inserts the value into the item tree from where it can be displayed in graphs or used in the plc program.

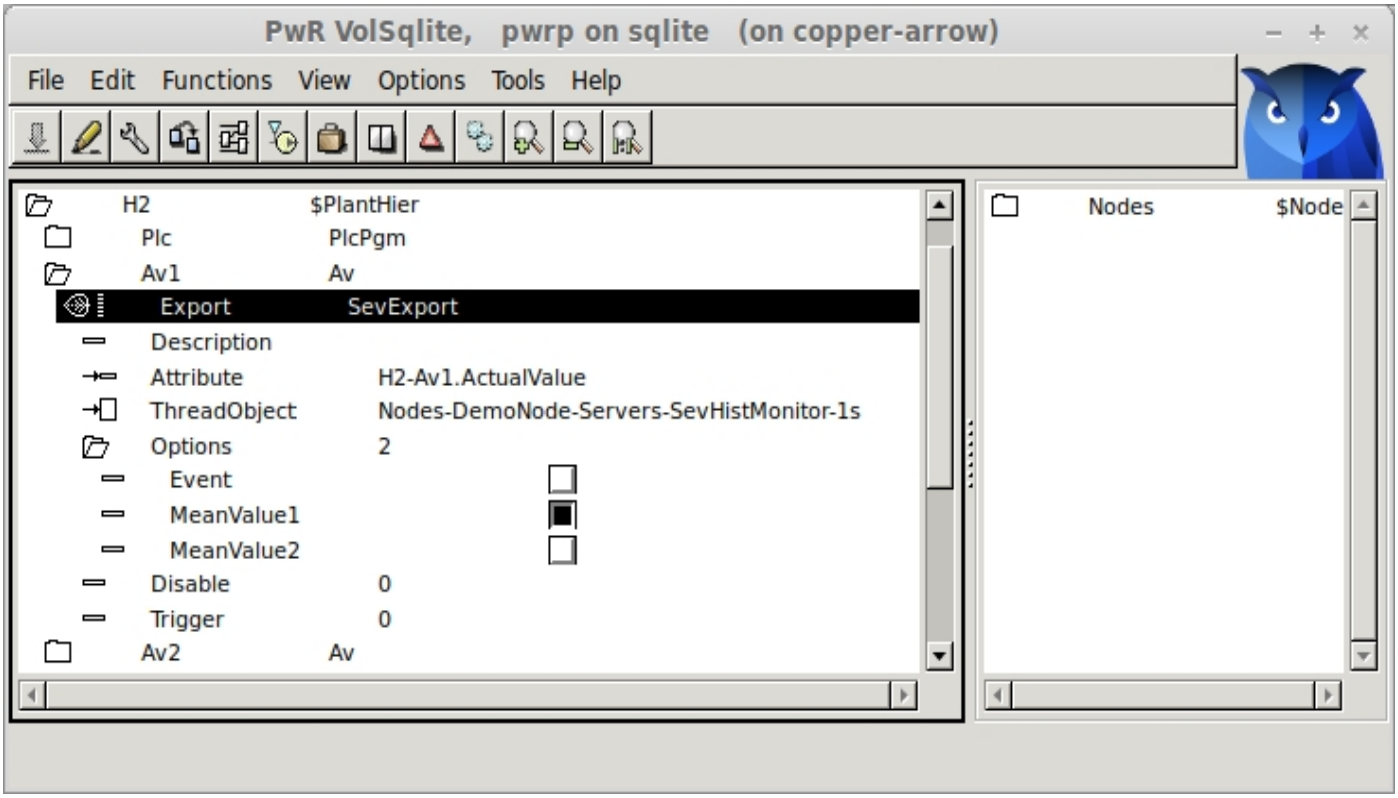


Fig Configuration of export with SevExport

Mean value calculation

Also for exported attributes, a mean value calculations can be configured by setting MeanValue1 or MeanValue2 in Options in the SevExport object.

The server makes the mean value calculation with two different times that are stated in the SevImportServer object, attributes MeanValueInterval1 and MeanValueInterval2. In SevExport.Options is stated which on of these times the mean value calculation should be made with.

The mean vaue is displayed in the item tree, and can be referred to from Ge graphs and applicaions with the suffix '.__MeanValue', eg 'pwrNode-sev-H2-Av2.ActualValue.__MeanValue'.

Event triggered export

By setting Event in Options in the SevHist object, it is possible to control when the storage is performed. When Trigger in the SevHist object is set to 1, the current value is sent to the server process. Trigger is reset automatically when the value is sent.

5 Internal database structure.

MariaDB/MySQL and sqlite are SQL databases where data is stored in tables. In HDF5 an hierarchy of groups are created where data is stored in datasets. The notation with tables below applies to SQL but the structure of the datasets in HDF5 is similar.

The database is named 'pwrp__'systemname'' and contains the tables 'items', 'objectitems', 'objectitemattributes', 'sev_stat' and 'sev_version'. Furthermore there is one table for each atttribute or object that is stored.

items

The items table contains information from all SevHist and SevHistEvents that it stored.

Name	Type	Description
id	integer	Identity.
tablename	varchar	Name of table with history data.
vid	integer	Volume identity of stored object.
oix	integer	Object index of stored object.
oname	string	Object name.
aname	string	Attribute name. If it's a SevHistEvent item, 'Events'.
uptime	datetime	Start time
cretime	datetime	Creation time for item.
storagetime	integer	Storage time in seconds. After this time the data will be deleted.
deadband	float	Deadband.
options	integer	Options.
scantime	float	Scan time.
description	string	Description of stored object.
vtype	integer	Attribute type.
vsize	integer	Attribute size in bytes.
unit	string	Attribute unit.

objectitems

The Objectitem table contains information about all SevHistObject that is stored.

Name	Type	Description
id	integer	Identity.
tablename	varchar	Name of table with history data.
vid	integer	Volume identity for stored object.
oix	integer	Object index for stored object.
oname	string	Object name.
aname	string	Not used.
uptime	datetime	Start time.
cretime	datetime	Creation time for item.
storagetime	integer	Storage time in seconds. After this tim the data will be deleted.
deadband	float	Deadband.
options	integer	Options.
scantime	float	Scan time.
description	string	Description of stored object.

objectitemattributes

To get a complete description of a SevHistObject, information of the attributes each object contains is needed in addition to the content of objectitems. This is stored in the objectitemattributes table with one row for each attribute.

Name	Type	Description
tablename	varchar	Name of table with history data
attributename	string	Attribute name.
attributeidx	integer	Attribute index.
attributetype	integer	Attribute type.
attributesize	integer	Attribute size in bytes.

History tables for individual attributes

Tables for storage of process values configured with SevHist objects.

Name	Type	Description
id	integer	Identity.
time	datetime or integer	Time.
ntime	integer	Nano seconds if high time resolution is configured.
value	arbitrary type	Process value.

History tables for whole objects

Tables for storage of process values for whole objects configured with SevHistObject objects.

Name	Type	Description
sev__id	integer	Identity.
sev__time	datetime or integer	Time.
sev__ntime	integer	Nano seconds if high time resolution is configured.
'attributename1'	arbitrary type	Process value for first attribute in the object.
'attributename2'	arbitrary type	Process value for second attribute in the object.
...		

History tables for alarm and events

Tables for storage of events configured with a SevHistEvent object.

Name	Type	Description
time	integer	Time.
ntime	integer	Nano seconds.
eventtype	integer	Event type.
eventprio	integer	Event priority.
eventid_nix	integer	Event identity, nix part.
eventid_birthtime	integer	Event identity, birthtime part.
eventid_idx	integer	Event identity, idx part.
supobject_vid	integer	Supervision object attrref, vid part.
supobject_oix	integer	Supervision object attrref, oix part.
supobject_offset	integer	Supervision object attrref, offset part.
supobject_size	integer	Supervision object attrref, size part.
eventtext	varchar	Event text.
eventname	varchar	Event name.
eventstatus	integer	Event status.

sev_stat

sev_stat contains statistics. Nowadays this information is also available in the SevServer object.

Name	Type	Description
current_load	float	Current storage load in percentage.
medium_load	float	Medium storage load in percentage.
storage_rate	float	Number of stored items per second.
medium_storage_rate	float	Medium value of number of stored items.
datastorage_msg_cnt	integer	Number of storage transactions since startup.
dataget_msg_cnt	integer	Number of history data requests since startup.
items_msg_cnt	integer	Number of item messages.
eventstore_msg_cnt	integer	Number of storage messages for alarms and events.

sev_version

Contains the current version of the sev databse. the Sev version is incremented when the database structure is modified and does not follow the version of ProviewR releases.

Name	Type	Description
version	integer	Current version.

6 Databases

The most used database is MariaDb, but there is also support for Sqlite and HDF5 with limited functionality.

6.1 MariaDB/MySQL

MariaDB is the most used storage database in ProviewR. It also has full functionality for deadband and server threads.

The configuration is made by setting Database to MySQL in the SevServer object. Furthermore installation and start of mariadb-server is required on the server node. The pwrp user also has to be created

For MariaDB

```
mysql
MariaDB> create user pwrp@localhost;
MariaDB> grant all privileges on *.* to pwrp@localhost;
```

For MySQL

```
mysql
mysql> grant all privileges on *.* to pwrp@localhost;
```

The recommended database engine is InnoDB that is default in later versions.

For small databases the standard configuration of MariaDB can be used, but for dedicated server nodes there are som settings that should be made in the file /etc/mysql/my.cnf.

innodb_file_per_table
From maintenance view it is an advantage to have each table in a separate file. Then disk space can be retrieved for deleted signals.

innodb_log_file_size
Transactions are first written into log files before they are inserted into the data files. With larger log files the writing to the database files can be optimized and done sequentially. On the other hand a recovery of the database will take longer time. The size of the log files normally should be increased from the default value.

innodb_buffer_pool_size
Memory that is not used by the operating system, applications or MariaDB should be allocated to the buffer pool. A calculation of the buffer pool size can look like this. Let's say we have 16 Gb memory, 2 Gb is used by the operating system, the innodb_log_file_size is 0.5 Gb and there should be space in the cache for this, ProviewR needs 0.5 Gb, leave 1 Gb to other and the remaining 12 Gb can be configured for the innodb_buffer_pool_size.

query_cache_type och query_cache_size
Tables are continuously modified so there is no reason to cache the result of requests. Set these to 0.

Example of configuration

```
[mysqld]
innodb_log_file_size    = 512M
innodb_buffer_pool_size = 12G
innodb_file_per_table   = 1
innodb_flush_method     = O_DIRECT
query_cache_type        = 0
query_cacne_size        = 0
```

Maintenance and troubleshooting

With the MariaDB client 'mysql', the database can be inspected and modified. The name of the database is 'pwrp__'systemname'', eg 'pwrp__test57'. Below is an example of how to look at the items table and data for an individual item.

```
> mysql
MariaDB> use pwrp__test57;

MariaDB> select oname,tablename from items;
+-----------------------------+--------------------------------+
| oname                       | tablename                      |
+-----------------------------+--------------------------------+
| VolTest57:H5-Av1            | O000_100_100_001_000000ab__1   |
| VolTest57:H5-Dv1            | O000_100_100_001_000000b7__2   |
| VolTest57:H5-Dv2            | O000_100_100_001_000000be__3   |
| VolTest57:H1-Av1            | O000_100_100_001_00000018__4   |
| VolTest57:H1-Av2            | O000_100_100_001_0000002e__5   |
| VolTest57:H1-Av3ÅÄÖ         | O000_100_100_001_0000009f__6   |
| VolTest57:H1-Dv1            | O000_100_100_001_00000056__7   |
| VolTest57:H1-Iv1            | O000_100_100_001_0000005d__8   |
| VolTest57:H1-Iv2            | O000_100_100_001_0000005e__9   |
| VolTest57:H1-Av4            | O000_100_100_001_0000013d__10  |
| VolTest57:H18-SevHistEvents | O000_100_100_001_00000ef3__11  |
| VolTest57:H1-Av5            | O000_100_100_001_00000f96__12  |
+-----------------------------+--------------------------------+

MariaDB> select time,value from O000_100_100_001_00000018__4 order by id desc limit 10
+---------------------+----------+
| time                | value    |
+---------------------+----------+
| 2021-07-14 14:21:32 |  83.7721 |
| 2021-07-14 14:21:26 |  97.9997 |
| 2021-07-14 14:21:22 |  99.8689 |
| 2021-07-14 14:21:18 |  95.4521 |
| 2021-07-14 14:21:13 |   81.552 |
| 2021-07-14 14:21:07 |  54.4918 |
| 2021-07-14 14:20:50 | -47.3081 |
| 2021-07-14 14:20:42 | -83.9247 |
| 2021-07-14 14:20:36 | -98.0545 |
| 2021-07-14 14:20:32 | -99.8544 |
```

```
| 2021-07-14 14:20:28 | -95.3687 |
| 2021-07-14 14:20:23 | -81.3886 |
| 2021-07-14 14:20:17 | -54.2567 |
| 2021-07-14 14:20:02 |  36.1479 |
| 2021-07-14 14:19:53 |  80.5102 |
| 2021-07-14 14:19:47 |  96.7008 |
| 2021-07-14 14:19:43 |  99.9982 |
| 2021-07-14 14:19:42 |  99.8383 |
| 2021-07-14 14:19:38 |  95.2808 |
| 2021-07-14 14:19:33 |  81.2193 |
+---------------------+----------+
```

6.2 Sqlite

Sqlite doesn't require installation of any further server process, however the functionality is limited. Support for server threads and deadband with linear regression is missing.

Sqlite is configured by setting Database in the SevServer object to Sqlite.

The database file is created in $pwrp_db with the name pwrp__'systemname'.dbsqlite, eg pwrp__test57.sqlite.

Maintenance and troubleshooting
The database can be examined with 'sqlite3'.

```
> sqlite3 $pwrp_db/pwrp___test57.dbsqlite
sqlite> select oname,tablename from items;
VolTest57:H1-Av1|O000_001_001_002_0000004a__0
VolTest57:H1-Av2|O000_001_001_002_0000004c__2
VolTest57:H1-Av3|O000_001_001_002_0000004e__3
VolTest57:H1-Av4|O000_001_001_002_00000050__4
VolTest57:H1-Dv1|O000_001_001_002_00000052__5
VolTest57:H1-Iv1|O000_001_001_002_00000054__6
VolTest57:H1-Iv2|O000_001_001_002_00000056__7

sqlite> select time,value from O000_001_001_002_0000004a__0 order by id desc limit 20;
2021-07-09 16:25:10|-26.695
2021-07-09 16:25:09|-20.593
2021-07-09 16:25:08|-14.409
2021-07-09 16:25:07|-8.16822
2021-07-09 16:25:06|-1.89594
2021-07-09 16:25:05|4.38459
2021-07-09 16:25:04|10.6471
2021-07-09 16:25:03|16.8675
2021-07-09 16:25:02|23.0222
2021-07-09 16:25:01|29.0861
2021-07-09 16:25:00|35.0345
2021-07-09 16:24:59|40.8447
2021-07-09 16:24:58|46.4945
2021-07-09 16:24:57|51.9609
2021-07-09 16:24:56|57.2216
2021-07-09 16:24:55|62.2567
```

```
2021-07-09 16:24:54|67.0468
2021-07-09 16:24:53|71.5718
2021-07-09 16:24:52|75.815
2021-07-09 16:24:51|79.7587
sqlite> .quit
```

6.3 HDF5

HDF5 doesn't require installation of any further server process, however the functionality is limited. Support for server threads and deadband with linear regression is missing.

HDF5 is configured by setting Database in the SevServer object to HDF5.

The database file is created on $pwrp_db with the name 'pwrp__'systemname'', eg pwrp__test57.hdf5.

The file contains the groups 'Dir' and 'Tables', where 'Dir' contains the datasets 'Cmn', 'Items', 'ObjectItems', 'ObjectItemAttributes' and 'Stat'. Under 'Tables' there is one dataset for each stored attribute or object, eg 'O000_001_001_003_0000004a__0', 'O000_001_001_003_0000004c__1' etc.

Maintenance and troubleshooting
It is possible to inspect the data file with Python by installing python3-h5py.

```
> python3
>>> import h5py
>>> f = h5py.File('/usr/pwrp/test57/src/db/pwrp__test57.h5','r')
>>> list(f.keys())
['Dir', 'Tables']
>>> list(f['Dir']['Items'])
[(0, b'O000_001_001_003_0000004a__0', 65795, 74, b'VolHdf5:H1-Av1', b'ActualValue',
0, 1625831988, 3600, 1., 76, 1., b'', 98306, 4, b'', 0),
(1, b'O000_001_001_003_0000004c__1', 65795, 76, b'VolHdf5:H1-Av2', b'ActualValue',
0, 1625831988, 3600, 1., 76, 1., b'', 98306, 4, b'', 0)
...
>>> list(f['Tables']['O000_001_001_003_0000004a__0']['Data'])
[(1625835966, 0, -93.87609), (1625835967, 0, -91.52528), (1625835968, 0, -88.812),
(1625835969, 0, -85.74749), (1625835970, 0, -82.343864), (1625835971, 0, -78.61555),
(1625835972, 0, -74.57546), (1625835973, 0, -70.24045), (1625835974, 0, -65.627655),
...
```

7 Extract history data

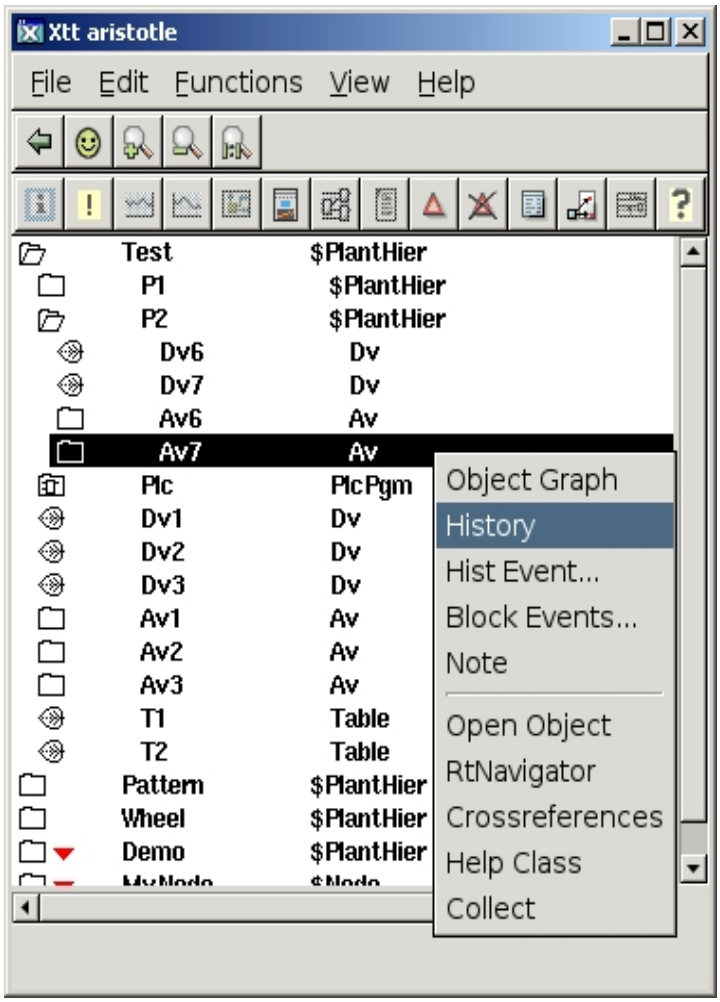
Here some examples are shown on how to extract an display history data.

7.1 Xtt

The History method in Xtt for an object will display a curve window with the process history. The History method is activated from

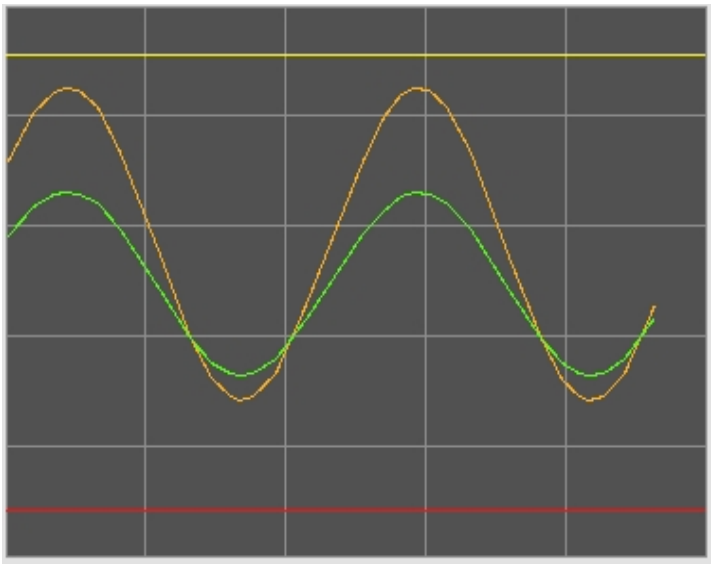
- the popup menu for the object.
- the tool panel in the object graph.
- the menu in the object graph.
- the Xtt command 'open history'.

The curve window can also be opened with the 'Open Graph' method for a SevHist object, or the 'Open Graph' method in the item tree.



7.2 Ge graph

The SevHist component in the Ge editor is found under Analog/SevHist in the palette. The component can display two history curves. It can be configured i two ways, either it's connected to SevHist objects or the object identity for the history is suppled.
If the graph should be opened from a sev server node the second alternative has to be used. If it's only going to be viewed on operator and process stations, the first alternative can be used.



7.3 Multivariate Analyser

The Multivariate Analyser can read history for a number of items and display curves, scatterplots, create models with linear regression and neural networks etc.

See the Multivariate Analyser chapter below.

7.4 Event Analyser

The Event Analyser read the history for alarms and events and display statistics and curves.

See the Alarm and Event Analyser chapter below.

7.5 Python

The ProviewR Python runtime module, pwrrt, contains functions to fetch history data from a sev server.

pwrrt can be execute on a node that has QCom contact to the history server.

Example
pwrrt.getSevItemData() fetches history for one attribute. It returns a tuple with three elements, number of samples, a tuple with values and a tuple with times.

```
import pandas as pd
from datetime import datetime
import pwrrt

pwrrt.init("appl")

result = pwrrt.getSevItemData( 'localhost', '_O0.254.254.204:68',
 'ActualValue', '00:02:00', 'now', 1000)
for i in range(result[0]):
  print(i, str(result[2][i])[:22], result[1][i])
```

Example
pwrrt.getSevItemsDataFrame() can request history for several attributes, and returns the history in a structure that can be inserted into a pandas frame. The first column contains the time, and the other columns the attribute values.

```
import pandas as pd
import pwrrt

pwrrt.init("appl")

oidlist = []
attrlist = []
isobjectlist = []

# Append first attribute
oidlist.append('_O0.254.254.204:68')
attrlist.append('ActualValue')
isobjectlist.append(0)

# Append second attribute
oidlist.append('_O0.254.254.204:69')
attrlist.append('ActualValue')
isobjectlist.append(0)

result = pwrrt.getSevItemsDataFrame( 'localhost', oidlist, attrlist,
 isobjectlist, '00:02:00', 'now', 0.5, 1000)
columns = ('time', 'A1', 'A2')
data = pd.DataFrame(data=result)
data.columns = columns
print(data)
```

7.6 Mqtt server

Mqtt server in ProviewR makes is possible to retrieve history data on any platform that has implemented the MQTT client.

The server replies to requests with different actions. The "history" action requests history data for an attribute, and the "eventhist" action request alarm and event history.

Example
This is a code example in Python with the MQTT client module python3-paho-mqtt.

History is fetched from the local MQTT server, topic 'proviewr/server'. The reply is requested to be sent to topic 'repl/history', and the request is to fetch history data for the attribute H1-Av1.ActualValue for the last 15 minutes.

```
#!/usr/bin/python3
#
import paho.mqtt.client as mqtt
import sys
import time
from datetime import datetime
import json
import matplotlib.pyplot as plt
from datetime import datetime

def on_message(client, userdata, message):
    data = json.loads(str(message.payload.decode("utf-8")))

    # Convert time strings to datetime objects
    t = []
    for dt in data['time']:
        t.append(datetime.strptime(dt+'0000', '%d-%b-%Y %H:%M:%S.%f'))

    # Plot the curve, use drawstyle='steps-pre' for digital signals
    plt.plot(t, data['values'], label='Diff')
    plt.show()


# Connect to MQTT server
client = mqtt.Client('Aristotle')
client.username_pw_set('pwrp','pwrp')
client.on_message = on_message
client.connect('localhost')

# Subscribe to reply
client.subscribe("repl/history", 1)

# Send history request
client.publish('proviewr/server', '{"action":"history",'  \\
'"reply":"repl/history","server":"localhost","object":"H1-Av1",'  \\
'"attribute":"ActualValue","from":"0:15:0","to":"now","maxrows":2000}')

for i in range (0, 3):
  print("Loop");
  client.loop_start()
  time.sleep(1)
  client.loop_stop()
```

8 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.

8.1 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 csv-file.

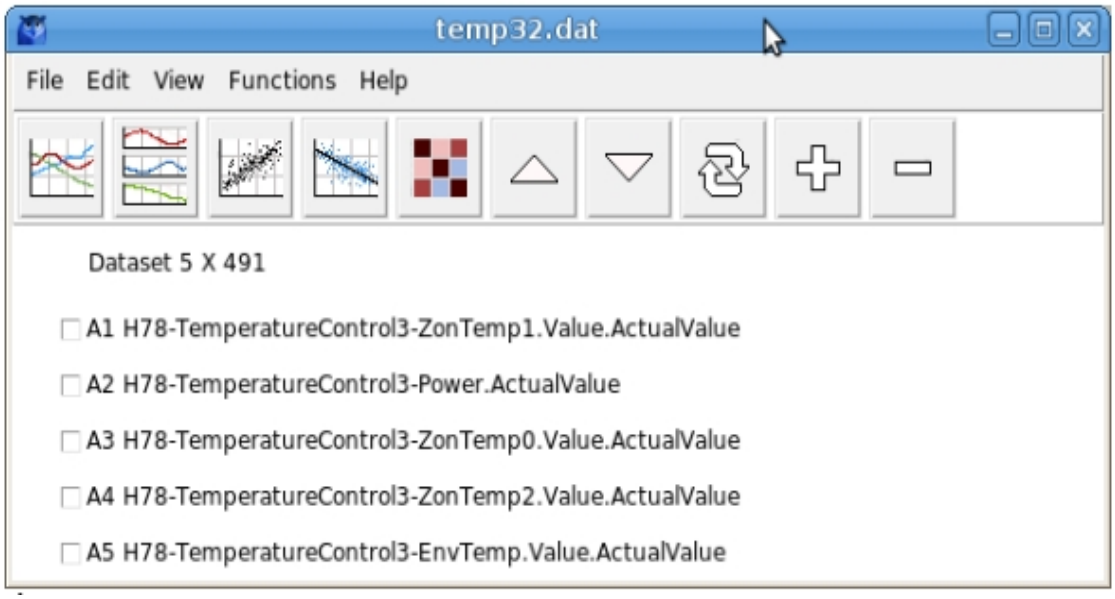


Fig Dataset

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.

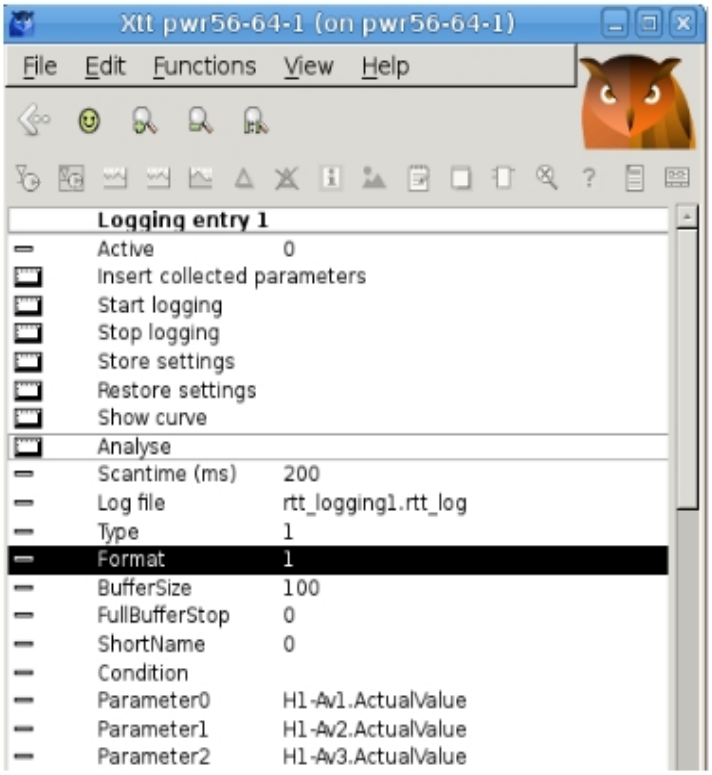


Fig Xtt logging

csv file

Data can be read from csv files with format displayed below. The first row is 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
...
```

8.2 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.

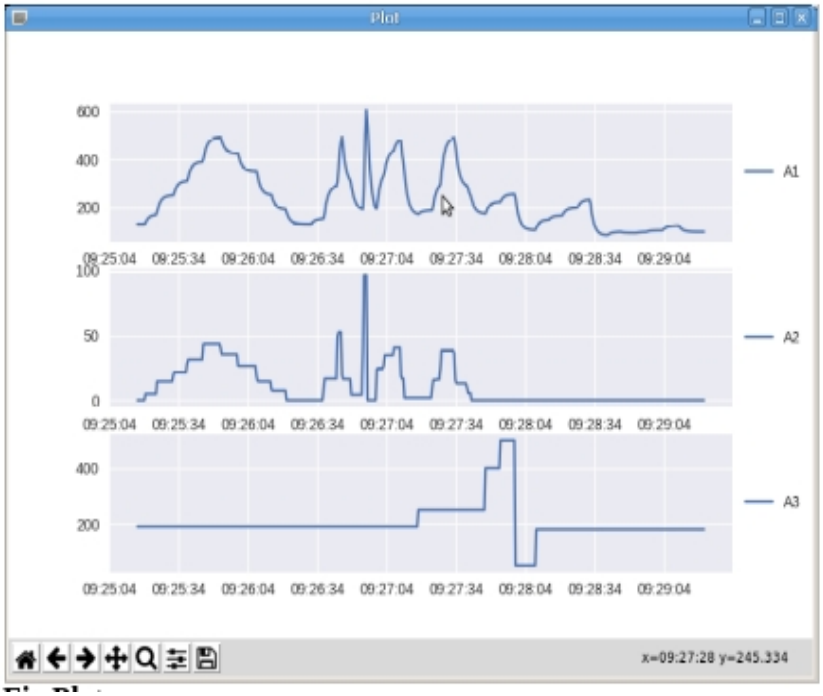


Fig Plot

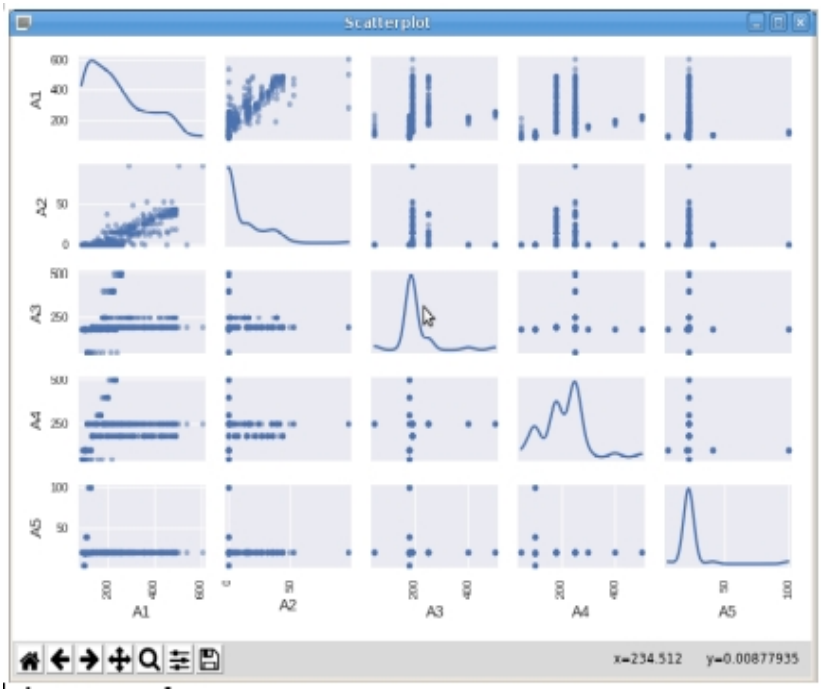


Fig Scatterplot

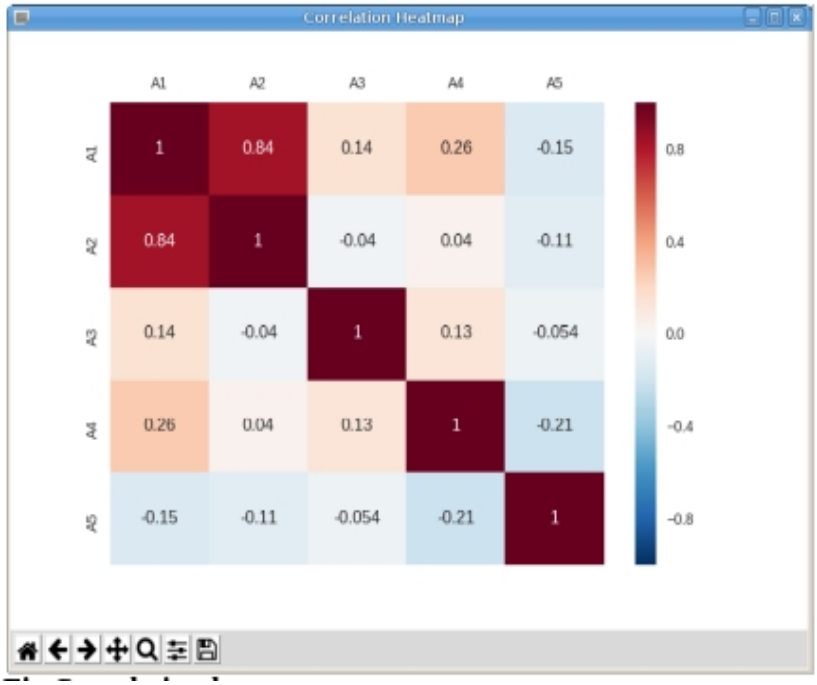


Fig Correlation heatmap

8.3 Edit data

The dataset can be edited 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.

8.4 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 csv 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 a

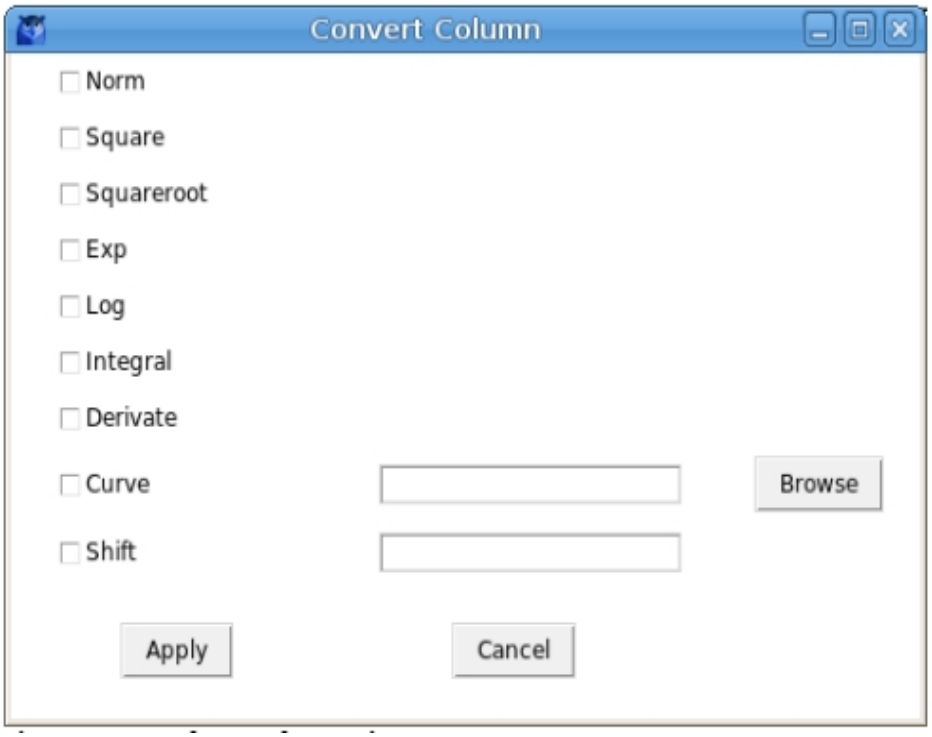


Fig Convert column alternatives

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 csv file.
- Constant. Will create a column where all rows has the specified value.
- Shift. Values in the column will be shifted forward or backward.

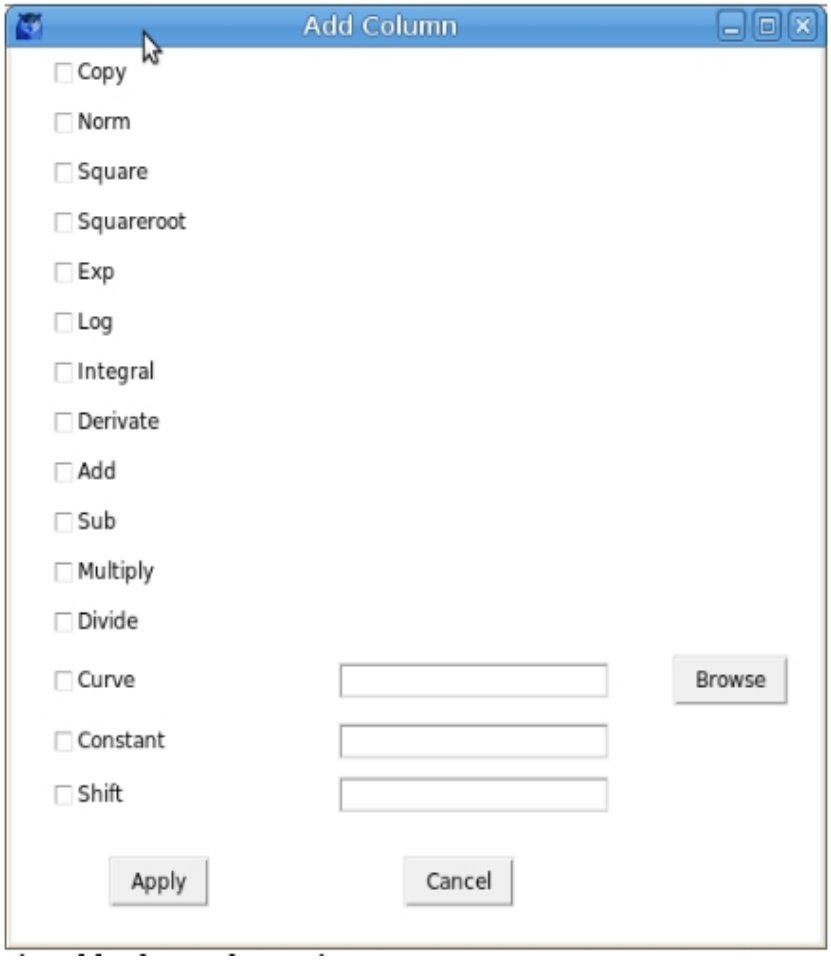


Fig Add column alternatives

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 formula is saved from 'File/Save Formula' in the menu, and applied from 'File/Apply Formula'.

8.5 Linear regression

Linear regression will create a model where one parameter, y, can be calculated from a number of input values x1 - xn. y is supposed to have linear dependencies of the input values, and the formula is

y = a0 + a1 * x1 + a2 * x2 + ... + an * xn

where a0, a1, ..., an 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 are also implemented.

Fig Linear regression

8.6 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.

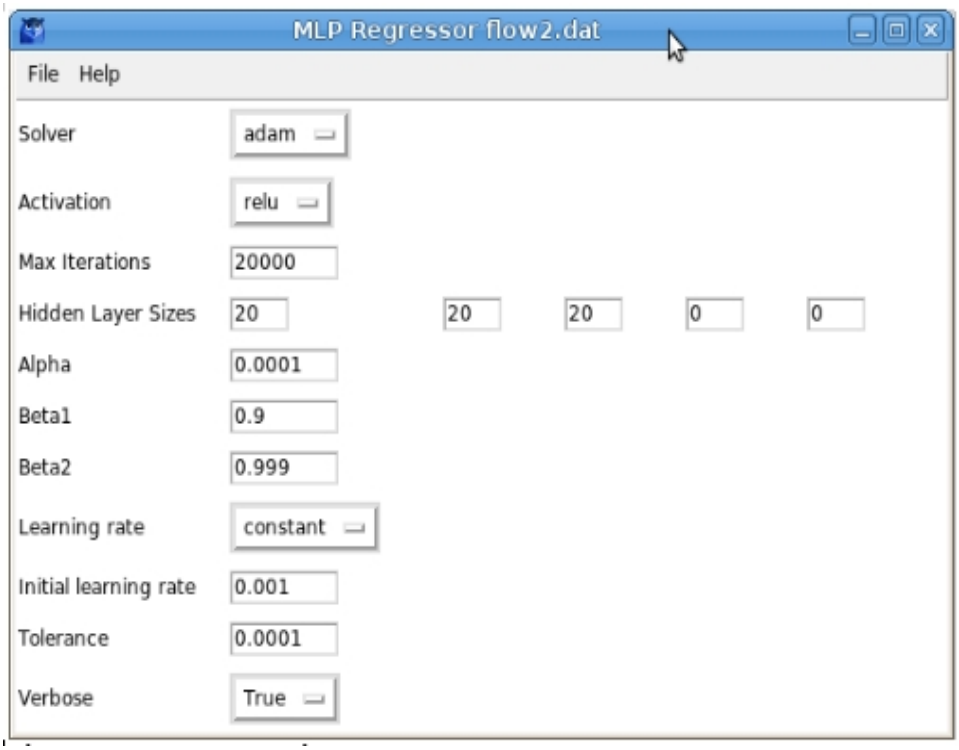


Fig MLP regressor settings

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

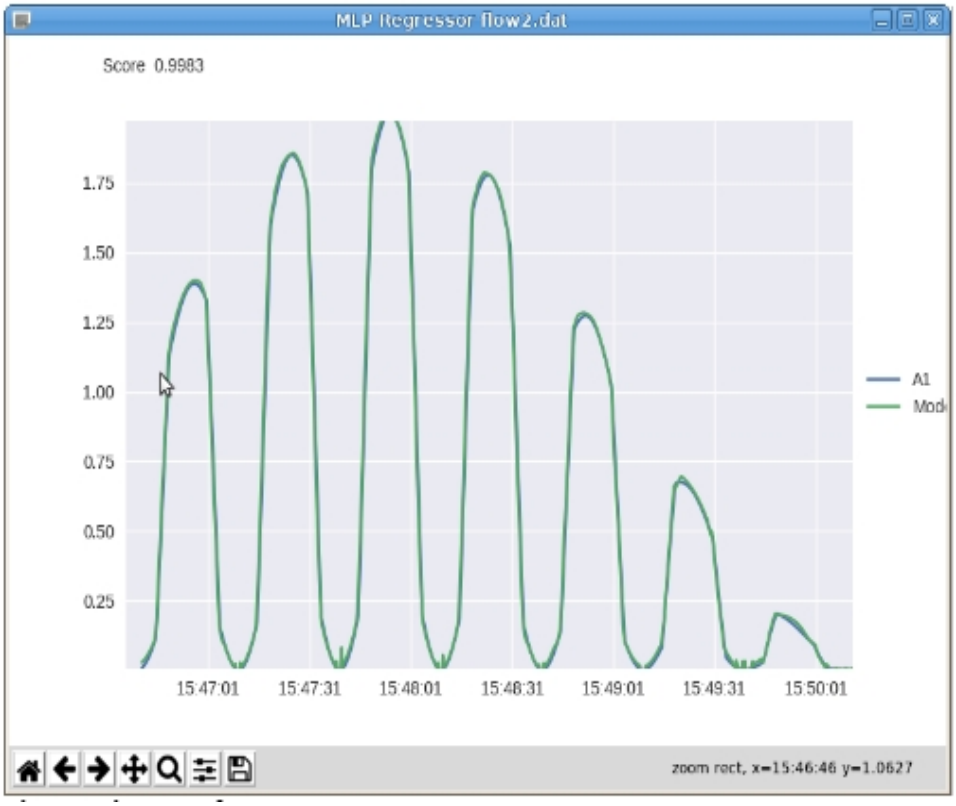


Fig Training result

9 Alarm and event analyser

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.

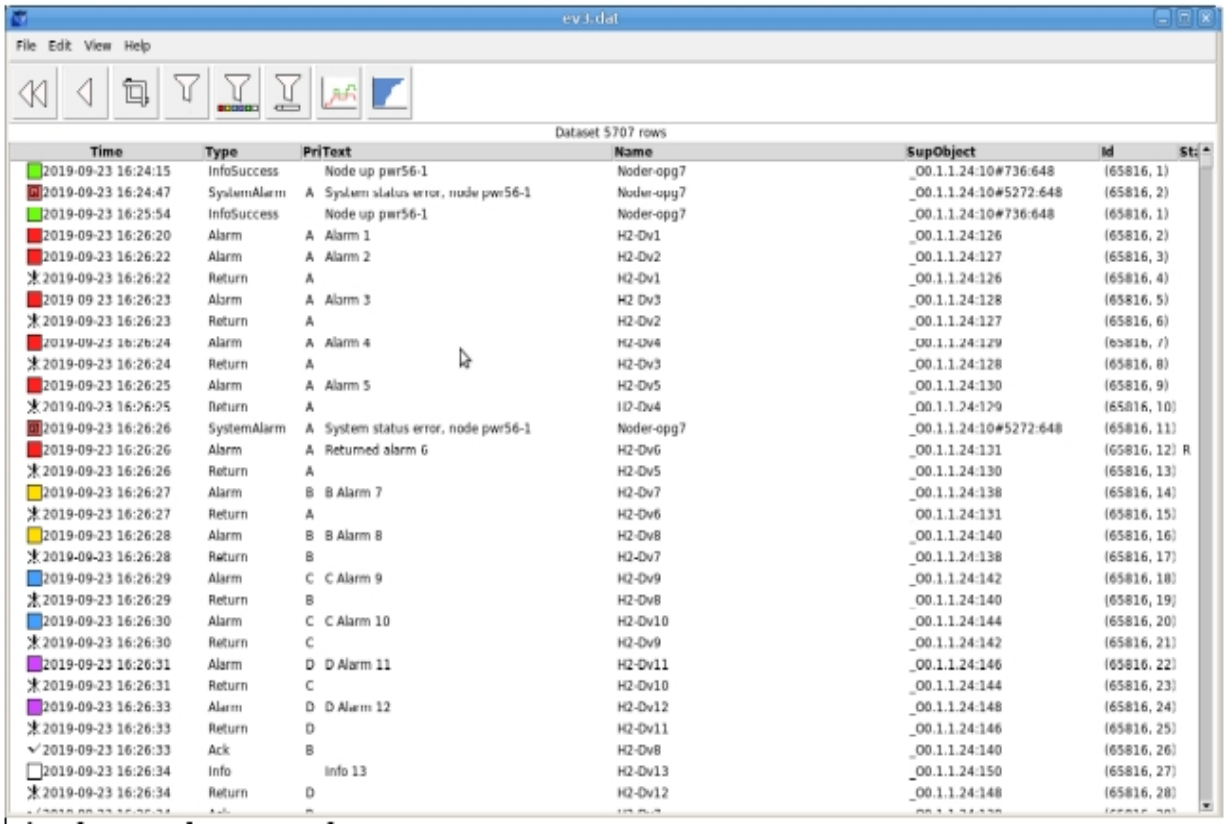


Fig Alarm and event analyser

Example of plots are the 'Event frequency histogram' that show the most frequent alarms, and 'Not returned alarms' that shows the number for concurrent alarms as a function time.

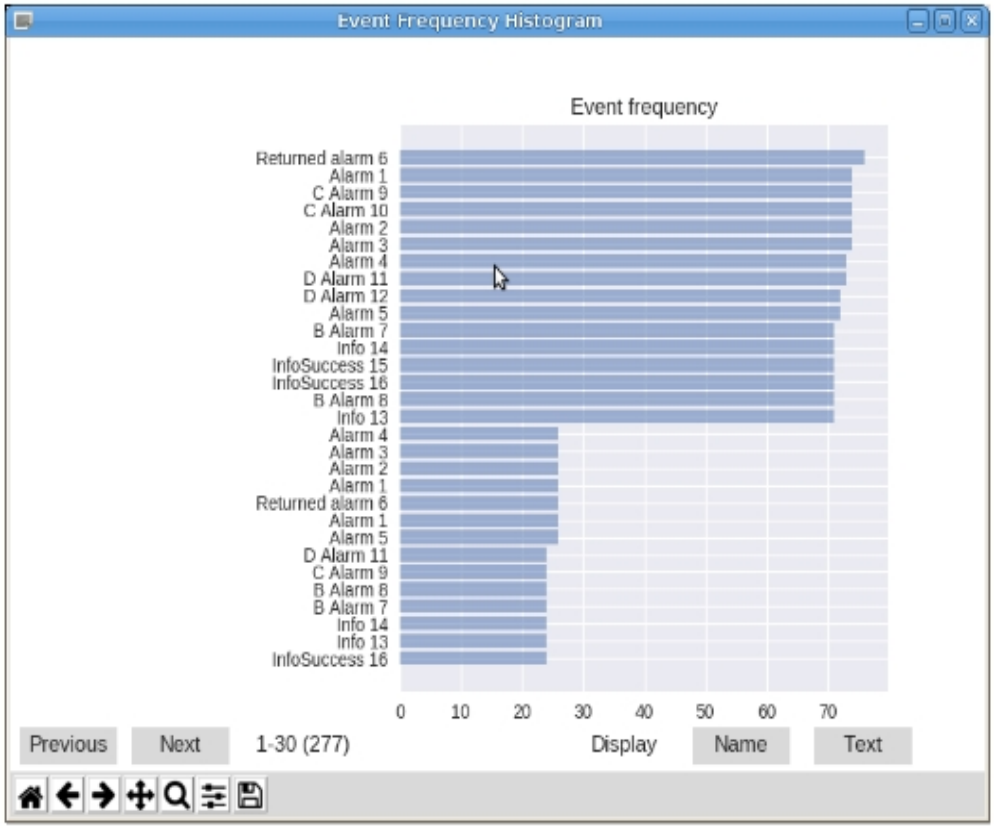


Fig Event frequency histogram

Fig Not return alarms

10 Storage station configuration

A storage station is generated as a process or operator station by installing the pwrrt package.

Communicaton with process stations

A storage station can serve a plant with several process stations that belongs to different projects with different version. For this reason the storage station normally doesn't have nethandler connections but QCom only connections.

This is configured by creating FriendNodeConfig objects for all nodes that the storage station should have contact with, and set QComOnly in Connection. If the storage station is placed in a project with process stations, you set QComAutoConnectDisable in the BusConfig object. Then the links between nodes in the projects are configured with FriendNodeConfig objects.

Upgrading

Note when upgrading sev station from versions before 5.8
- The pwrsev package is discontinued since V5.8.0 and the pwrrt package should be used instead.
- The node should be configured with a NodeConfig object instead of SevNodeConfig.
- The root volume should be configured with a RootVolumeConfig object and edited.