Release Notes V4.6

2008 11 04

Table of Contents

Upgrading to Proview V4.6.0

This document describes new functions i Proview V4.6.0, and how to upgrade a project from V4.5.0 to V4.6.0.

New functions

Fast scan times

Handling of the plc-threads is changed to make it possible to run with very fast scan times (< 1 ms). To run with fast cycle times on Linux you have to run a kernel with Linux RT-Preempt patchset ("real time linux"). We have successfully tested with linux-2.6.24.7-rt17 (see for example www.osadl.org for more info). Tickless kernel and HR Timers has to be switched on in the config-file for the kernel. Of course to make use of such fast cycle times you also need an I/O-system that is as fast.

Modbus TCP

Modbus/TCP has enhanced functionality and supports one new function code - "Write single coil" has been added. A lot more can be tuned now to get the best function out of yout Modbus I/O.

There is one new class for Modbus, the "Modbus_Master". All the slaves should be configured as children to a Modbus_Master-object.

Sev Storage Environment

Sev is the storage environment, where historical data is stored in a database. It is a complement to the other environments in Proview, the development, runtime and operator environment. Sev contains server processes that handles fetching and storage of of historical data. In the future, it will also handle functions for safe backup of attributes. Sev can be installed as a separate unit on a storage station, but it is also included in the runtime package and can be started in the runtime environment.

Storage Station

A Storage station is a server node where the sev package (pwrsev) is installed. This node communicates via Qcom with client nodes and receives historical data which is stored in a mysql database. The historical data can be handled and displayed on the storage station by the program sev_xtt, or it can be sent to the client nodes and viewed by rt_xtt in the operator environment.

Configuration

The configuration of a storage station is made in the Directory Volume, by creating a SevNodeConfig object below a BusConfig object. The node can be placed in a project with process and operator stations, or it be be placed in a separate project with the storage station alone. If the storage station will communicate with client nodes in other projects, they should be configured as

friend nodes, with FriendNodeConfig objects. The storage station should also be configured as a friend node to the client nodes. Note the the FriendNodeConfig object has a new attribute, Connection, that should be set to QcomOnly, in the FriendNodeConfig objects on both sides.

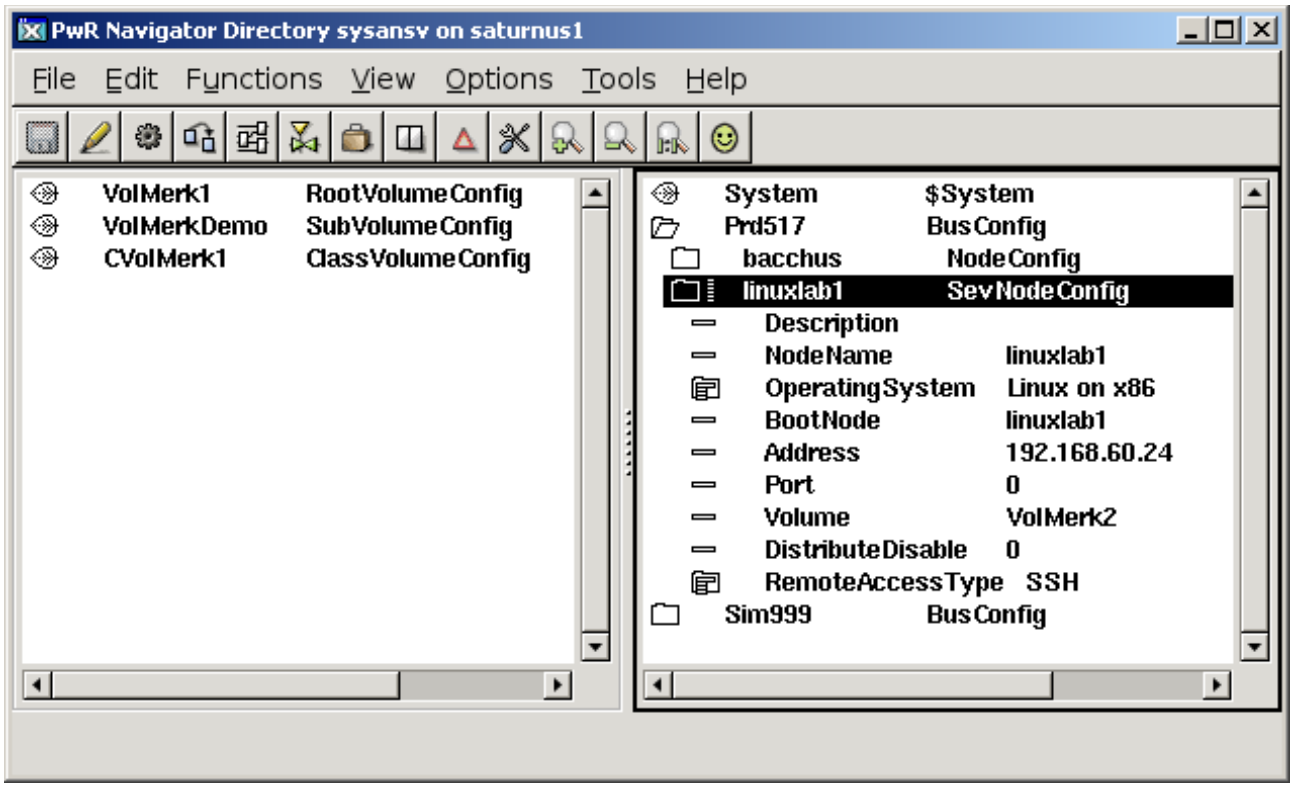


Fig Configuration of a Storage Station

Historical Data Storage

The historical data stoarge is devided int a server process, sev_server that handles the storage of the historica data, and a client process, rt_sevhistmon, that collects the data on the client node, and sends it to the server. The server process is a part of the storage environment and the client process a part of the runtime environment. The server projecess is also included in the runtime enviroment, which makes is possible to create a combined process/storage station.

Configuration

SevHist

Attributes that are to be stored in the historical database are configured with SevHist objects. The SevHist objects are placed in a root or sub volume, and the attribute is stated in Attribute in the SevHist object. The SevHist object is normally placed below the object that is to be stored, and if this object has an ActualValue attribute, this is automatically inserted into the SevHist object.

Attributes

Attribute	*Type*	*Desription*
Description	pwr_tString80	Optional description.
Attribute	pwr_AttrRef	Attribute that is to be stored.
ThreadObject	pwr_tObjid	Thread object that determines the storage frequency and the server node.

Attribute	*Type*	*Desription*
StorageTime	pwr_tDeltaTime	States how long the data is to be stored.
DeadBand	pwr_tFloat32	The size of the deadband.
Options	pwr_tMask	Options mask with the following bits: PosixTime 1: The time is stored as a posix time. 0: The time is stored as a sql time. HighTimeResolution: The time is stored with nano second resolution (otherwise 1 second resolution). ReadOptimized: Optimized for fast reading. UseDeadBand: Deadband is activated.

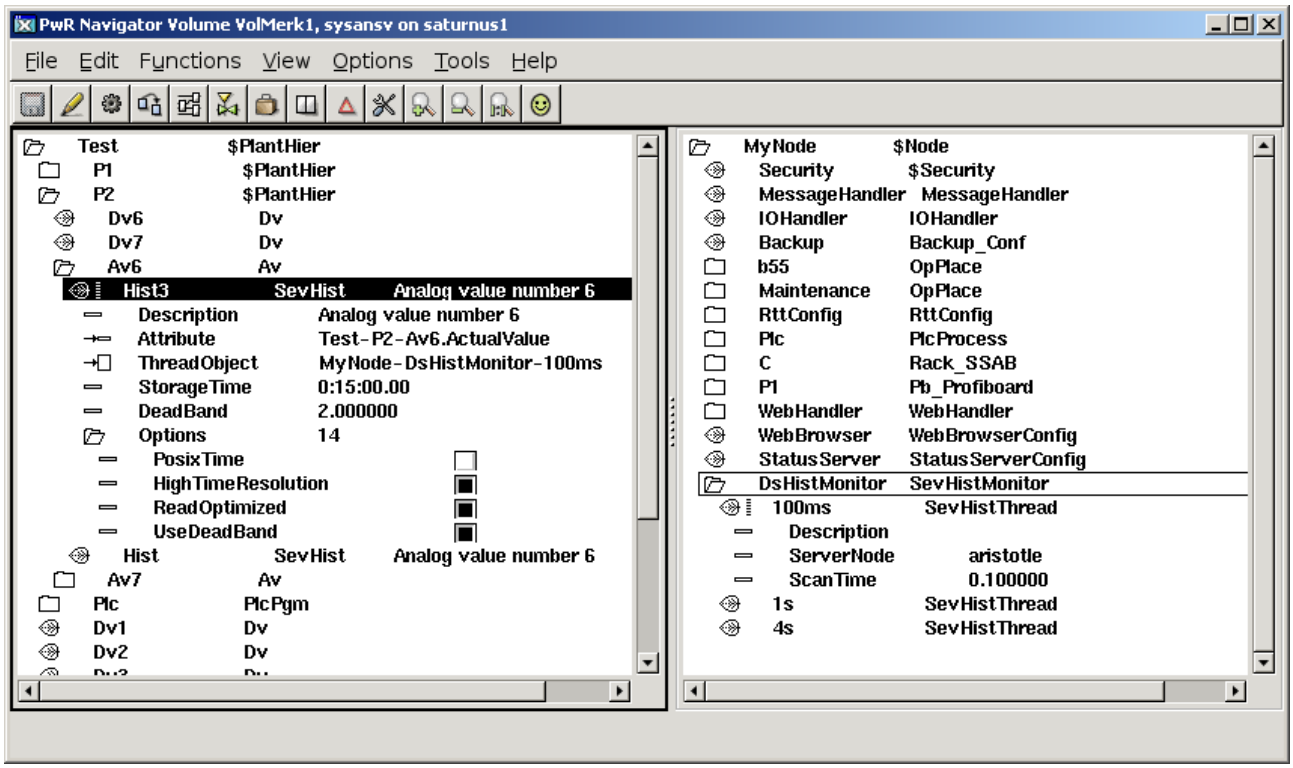


Fig Configuraion in a rootvolume with SevHist, DsHistMonitor and SevHistThread objects

SevHistMonitor

The client process, rt_sevhistmon, is configured with a SevHistMonitor object in the node hierarchy. rt_sevhistmon scans on DsHist objects and send historical data to the sev server.

SevHistThread

SevHistThread object are placed under the SevHistMonitor object to configure time bases and server nodes in rt_sevhistmon. Each SevHist object is connected to a SevHistThread object which determines to which server node, and how often data is sent.

Client API

The client API, sevcli, is used to fetch historical data from the database.

sevcli_init()

```
int sevcli_init( pwr_tStatus *sts, sevcli_tCtx *ctx);
```

Initialization of the client. Creates a context and opens a Qcom queue for the communication.

sevcli_close()

```
int sevcli_close( pwr_tStatus *sts, sevcli_tCtx ctx);
```

Is called to close the communication. Removes the Qcom queue and frees the context.

sevcli_set_servernid()

```
void sevcli_set_servernid( sevcli_tCtx ctx, pwr_tNid nid);
```

Is called to state which server node you want to communicate with. The server is stated with node identity.

sevcli_set_servernode()

```
int sevcli_set_servernode( pwr_tStatus *sts, sevcli_tCtx ctx, char *nodename);
```

Is called to state which server node you want to communicate with. The server is stated with node name.

sevcli_get_itemlist()

```
int sevcli_get_itemlist( pwr_tStatus *sts, sevcli_tCtx ctx,
                  sevcli_sHistItem **list, unsigned int *cnt);
```

Fetches a list with the stored attributes of the server.

sevcli_get_itemdata()

```
int sevcli_get_itemdata( pwr_tStatus *sts, sevcli_tCtx ctx, pwr_tOid oid,
                 char *aname, pwr_tTime starttime, pwr_tTime endtime,
                 int numpoints, pwr_tTime **tbuf, void **vbuf, int *rows,
                 pwr_eType *vtype, unsigned int *vsize);
```

Fetches the stored historical data for an attribute.

sevcli_delete_item()

```
int sevcli_delete_item( pwr_tStatus *sts, sevcli_tCtx ctx, pwr_tOid oid,
                 char *aname);
```

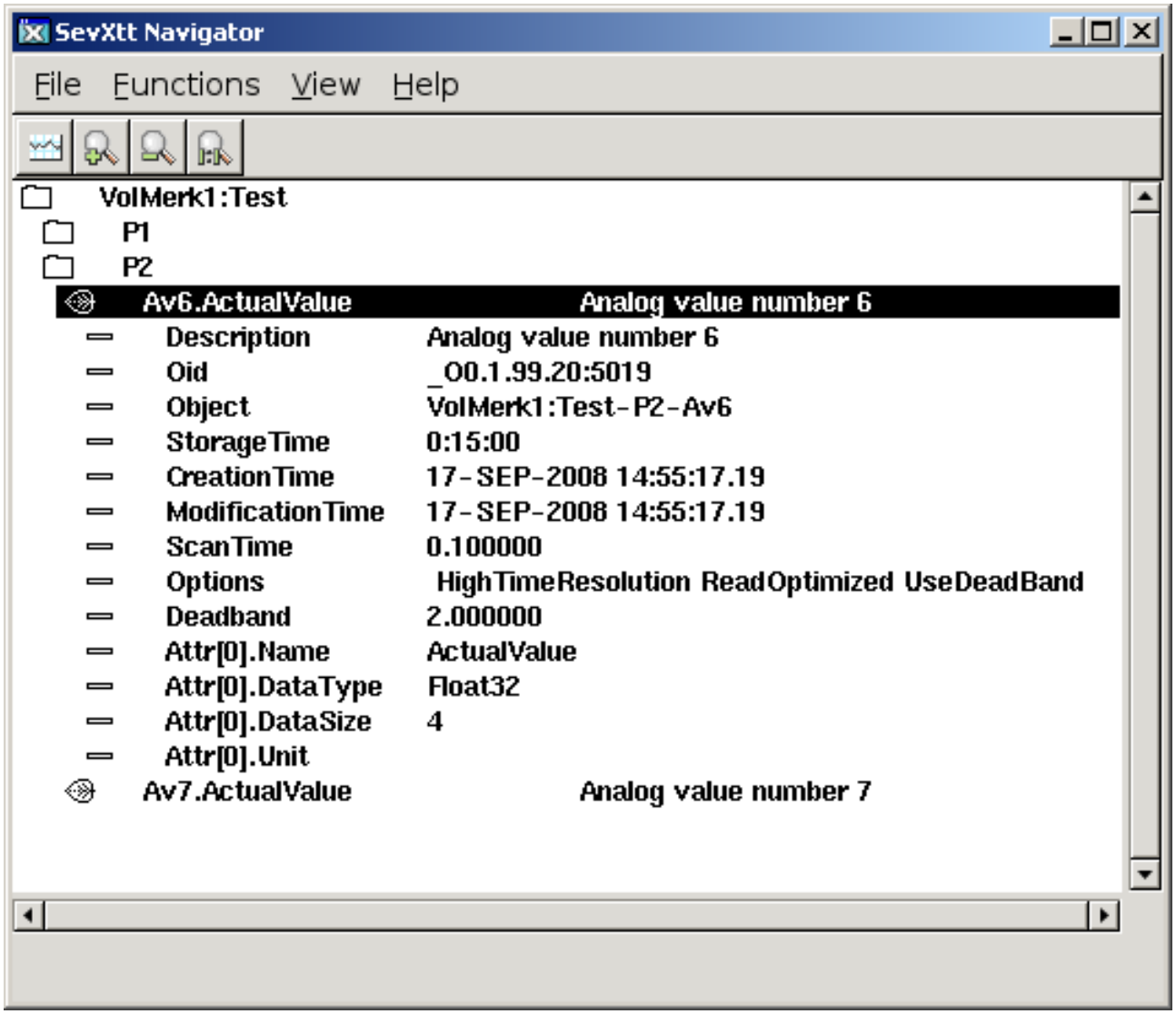
Removes all data for a stored attribute in the storage database.

Server process

The server process, sev_server, handles the database that stores the historica data, and communicates with the clients that supplies data, or request data to, for example, display in a curve window. The server can be started in the runtime environment, and is the configured by a SevServer object in the node hierarchy. In the storage environment, it is started automatically at Proview startup.

sev_xtt

Sev_xtt is a tool to navigate and display data in the historical database. It is started on the server node and can only view data in this node. At start, a list of all stored attributes is fetched and viewed in a tree structure. By opening an item (a stored attribute) the properties for the item is viewed.



An item viewed in Sev_xtt

If you select an item and activate the curve button, stored values are fetched and viewed in a curve window.

Old items can be removed from the data base from sev_xtt by selecting the item and activate 'Function/Delete Item' in the menu. Only users with the SevAdmin privilege are authorized to do this.

Access

There are two privileges that controls the access in sev_xtt: SevRead is required to view the content of the storage database, and SevAdmin is required to affect the storeage database (so far to delete storage items). There are two ways to gain these privileges:

1. To login as a user that is granted the privileges SevRead or SevAdmin.
2. To set a defaut privilege for sev_xtt in /etc/profile.cnf. The parameter sevXttDefaultPriv can have the values Read, Admin or None. If the value is Read, sev_xtt will have the SevRead privilege as default, if the value is Admin, sev_xtt will have the SevAmin privilge as default, and if the value is None, login is required.

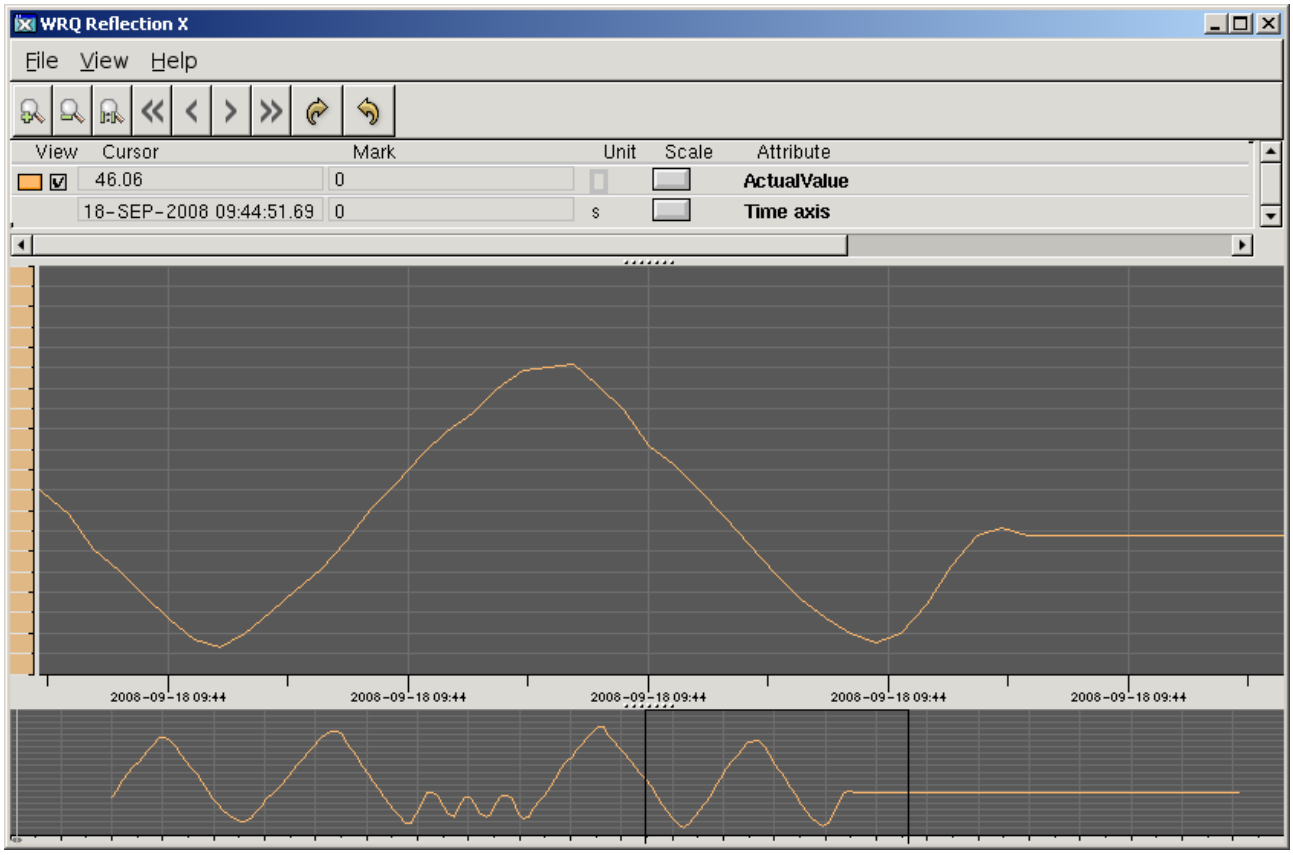


Fig Curve window in sev_xtt

The whole stored time intervall is fetched in about 500 points. If a higher resolution is wanted, an intervall is zoomed in and the right button in the tool panel is activated. A new message with 500 points in the marked time intervall is fetched and the curve for these is displayed. You can also decrease the resolution and increase the time intervall with the left arrow button.

rt_xtt

Historical data can also be viewed by rt_xtt in the operator environment. For objects that has a SevHist object as a child, a 'History' menu item is added to the popup menu. The history method views the same curve window as sev_xtt.

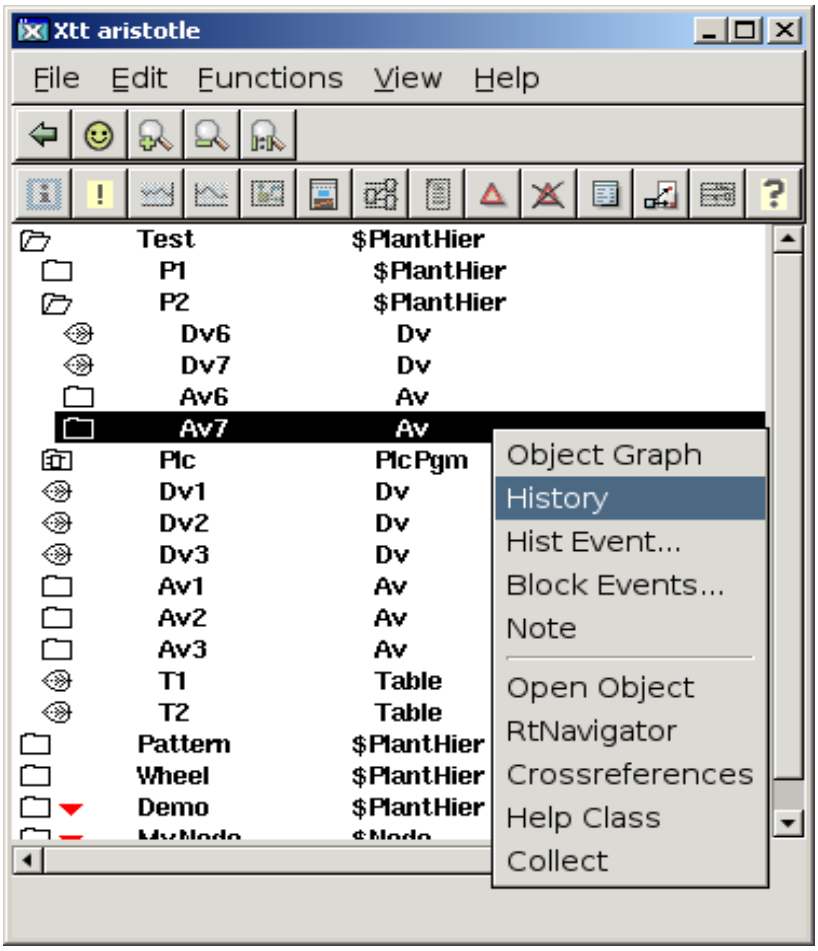


Fig History method in the operator environment

Security

Some new security functions has been added to 4.6.

New user database

The format and name of the user database is changed. The name is changed to $pwra_db/pwr_user2.dat. A new cryptation algorithm for password is used, and there are also attributes for email, sms, an phone. An identity will make it possible to reference a specific user in the future, and, for example, send an email or sms to a user when an alarm is triggered.

Security Object

Security settings are gathered into a Security object of class $Security. This should be configured in the node hierarchy under the node object. The attribute values of the security object can not be changed in runtime. The content of the object is fetched with the function gdh_GetSecurityInfo(). Functions that returns a pointer to an object will not work on this object.

rt_xtt privileges

When rt_xtt is started, the security object is used to configure the default privileges.

- if rt_xtt is started with an OpPlace object as an argument, the privileges are fetched for the corresponding User object. This function is unchanged, though a new thing is that the username

in the User object can't be changed in runtime.

- If rt_xtt is started without an OpPlace object, the default privileges are fetched from the the DefaultXttPriv attribute in the security object.
- You can also specify that a Proview user with the same name as the current Linux user determines the privileges. In this case the XttUseOpsysUser attribute in the security object should be set to 1.
- If no valid user is found, or the found user lacks valid runtime privileges, the login window is opened at rt_xtt startup.

rt_rtt privileges

The login procedure in rt_rtt is equal to the one in rt_xtt and ruled by the same attributes in the Security object. In the login picture, a proview user has to be specified, the rtt users in previous versions are not valid any more.

web privileges

Also the privileges in the web interface are affected by the Security object.

- A specific system group for the web interface can be specified in the WebSystemGroup attribute in the Security object. Only users in this group are able to login in the web interface. If no system group is specified, the ordinary system group for the project is used.
- The default privileges for the web interface are fetched from the DefaultWebPriv attribute in the Security object. If no privileges are specified here, login is required for any access.

Distributor

In prevous versions, the distributor has used ftp and rsh to copy and distribute files. This requires that the password for the pwrp user on operator and process stations is unchanged.

In V4.6 the distributor uses ssh and scp instead. For smooth operating, you need to create a public/private key pair, and copy the public key to the pwrp user on the process station.

Create the keys with

> ssh-keygen -t rsa

Copy the public key to the process/operator station and add to the file /home/pwrp/.ssh/authorized_keys

You can also keep public keys in $pwra_db/authorized_keys, and set the AuthorizedKeysFile bit in the distribute object in the directory volume. Then the distributor will copy this file to the process station and public keys for new users and development stations only need to be updated in one file.

The old ftp/rsh access is still available and which one, ftp/rsh or ssh/scp, the distributor should use is configured in the RemoteAccessType attribute in the NodeConfig object.

Crossreferences

The algorithm for searching for crossreferences when creating crossreferencefiles is improved. The format of the crossreference files is also changed and now also works for attributes in component classes.

The wtt command to create crossreference files is 'create crossreferencefiles' which replaces the previous 'create rttfiles'.

Ge journal file

Undo and redo function is added in Ge. A journal file stores all editing actions, and this also makes it possible to recover a terminated Ge session.

Area object size dependent on number of signals

The area objects, in which the values for signals are stored, prevously was created with a fix size. This implied that there was a maximum number of signals in a node (6000 for Di and 4000 for other signals). This size of the area objects are now dependent on the number of signals, i.e. the number of signals are unlimited.

Another new feature is that the Value attribute in the area objects, that keeps the signals values, now can be opened in rt_xtt, previouly only the first element was displayed.

Data inputs to function objects

The new plc object GetDataInput makes it possible to define data inputs in a function object with template plc code.

Keyboard editing

New functionallity has been added to the Configurator and the Plc editor to be able to edit entirely for the keyboard. The new accelerators have been choosen to be as backward compatible as possible as there are a many users that switch between different Proview versions. This is the reason Ctrl/D is used to create objects, which under normal conditions probably would do the opposite.

Configurator

Select objects

Select objects with the arrow keys Up and Down.

Create object

To create an object from the keyboard:

- set input focus to the palette with the TAB key.
- select the desired class.
- set input focus to the node or plant hierarchy with the TAB key. The selection in the palette is still active and marked with gray.
- Select the destination object (sibling or parent) to the object that should be created in the object tree.
- Press Ctrl/D to create the object as next sibling to the selected object, or Shift/Ctrl/D to create the object as first child to the selected object.

Move object

Select the object that should be moved and use Shift/Ctrl/Up to move the object upwards, and Shift/Ctrl/Down to move the object downwards. Shift/Ctrl/Right will move the object down in the object tree (child to the provious object), and Shift/Ctrl/Left will move the object up in the object tree.

Note that some objects are not valid as top objects in the object three and can not be moved from one top branch to another with the arrow keys as they have to pass a top location. In this case you have to use the mouse.

Delete object

Select one or several objects an press the Delete key.

Plc Editor

Scroll

Scroll with Alt/Up, Alt/Down, Alt/Left and Alt/Right.

Select an object

Select a function object with Shift/Up, Shift/Down, Shift/Right and Shift/Left.

Create an object

- Place the cursor inside the working area of the plc editor.
- Select a class in the palette with the arrow keys.
- Press Alt/D to create the object. The object is created at the cursor position. The reason for this is that if the cursor is move the object follows the cursor and the position is nailed when MB1 is clicked. In this case, though, you should move the object with Shift/Ctrl/Up, Shift/Ctrl/Down, Shift/Ctrl/Left and Shift/Ctrl/Right to the desired position.

Note that an object also can be created together with a connection, se create connection below.

Move objects

Select one or several objects and move them with Shift/Ctrl/Up, Shift/Ctrl/Down, Shift/Ctrl/Left and Shift/Ctrl/Right.

Delete objects

Select one or several objects and press the Delete key.

Create a connection

To create a connection between to function objecs you need to select two connectionpoints.

- Select the first connectionpoint with Ctrl/Up, Ctrl/Down, Ctrl/Left and Ctrl/Right.
- Lock this point with Ctrl ???
- Select the second connectionpoint with Ctrl/Up, Ctrl/Down, Ctrl/Left and Ctrl/Right.

– Create the connection with Ctrl/D.

If only one connectionpoint is selected, and Ctrl/D is pressed, the autocreate function is activated, i.e. a connection with a Get or Sto object is created. In Grafcet sequences objects following the Grafcet standard i created in the same way.

Connect

Connect a Get or Sto object to a signal with Ctrl/Q.

Note also that Expand/Compress object is moved to Ctrl/J and Ctrl/K.

DetachedClassVolume

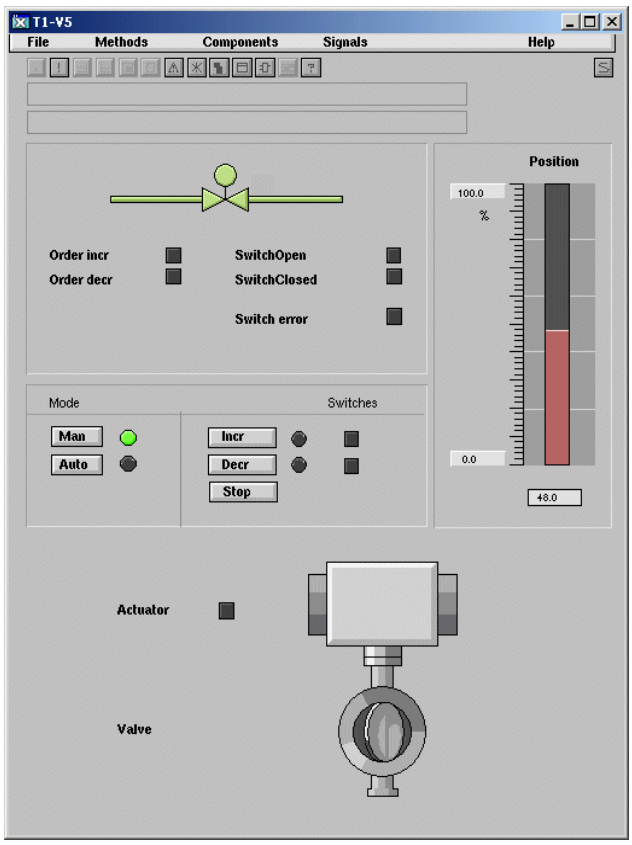
DetachedClassVolume is a new type of class volume that does not have the same tight dependency on other classvolume versions as an ordinary class volume.

Create Flow-files command

Functionality to create flow-files (files used for plc trace) for a PlcPgm or for all PlcPgm's in an hierarchy is added (command '`create flowfiles /hierarchy=`' in wtt navigator).

Valve with increment/decrement orders

A new basecomponent, BaseValveIncrDecr for a valve with digital outputs for increment and decrement orders. Threre are digital inputsignals for switch open and switch closed, and an analog signal for position feeback. All input signals are optional.



New Types

pwrb:TrueFalseEnum

Enumeration type with the value False (0) and True (1).

pwrb:YesNoEnum

Enumeration type with the value No (0) and Yes (1).

pwrb:SevHistOptionsMask

Mask type for the Options attribute in the SevHist object.

pwrb:NodeConnectionEnum

Enumeration type for connection level between friendnodes with the values Full and QcomOnly.

New Classes

$Security

Security settings are gathered into this object. It should be configured in the node hierarchy under the node object. The attribute values of the security object can not be changed in runtime. The content of the object is fetched with the function gdh_GetSecurityInfo(). Functions that returns a pointer to an object will not work on this object.

DetachedClassVolumeConfig

Configures a DetachedClassVolume in the directory volume.

DetachedClassVolumeLoad

Specifies that a DetachedClassVolume should be loaded at Proview startup.

SevNodeConfig

Configuration of a Storage station in the directory volume.

SevHistMonitor

Configuration of the rt_sevhistmon process, which supervises SevHist objects and sends historical data to the storage server. Reside in the node hierarchy.

SevHistThread

Specifies a time base and and a storage server node for the sevhist monitor. Positioned under a SevHistMonitor object.

SevHist

Specifies that an attribute should be stored in a historical database.

SevServer

Configuration object for the sev_server. This object is created in the node hierarchy and used to start the sev_server in the runtime environment. On a storage station, no configuration of the sev_server is needed.

GetDataInput

A plc object that makes it possible to define data inputs in a function object with template plc code.

Modbus_Master

Configuration object for Modbus/TCP I/O. All the Mobus_Slaves should be configured as children to this object.

BaseValveIncrDecr

Basecomponent for a vavle with digital increment and decrement output signals.

BaseValveIncrDecrFo

Function object to BaseValveIncrDecr.

BaseValveIncrDecrSim

Simulate object to BaseValveIncrDecr.

BaseAcuatorIncrDecr

Basecomponent for an actuator with digital increment and decrement output signals.

BaseAcuatorIncrDecrFo

Function object to BaseActuatorIncrDecr.

Modified classes

$ClassDef

Bit RtReadOnly added to Flags attribut to inhibit write in instances of the class at runtime.

$System

Flags Const set to all attributes, i.e. object can't be changed at runtime.

$ClassVolume

Attribute DvVersion added. This attribute will contain the version for detached classvolumes dependency.

UserReg

Attributes for email, phone and sms added.

User

Flag Const set to UserName attribute, i.e. attribute is not changeable at runtime. Attribute Password removed.

Distribute

Bit AuthorizedKeysFile added to Component mask.

NodeConfig

Attribute RemoveAccessType added. This attribute controls whether ssh/scp or ftp/rsh is used to access the node by the distributor.

FriendNodeConfig

Attribute Connection added. Configures the connection level to the friend node, and can have the value QcomOnly or Full.

Modbus_TCP_Slave

Changes as follows below:

ReconnectLimit

Has no meaning anymore, kept for possible future use.

ResponseTime

New. Defines the maximum time to wait for a response from a slave.

SingleOp

New. Defines whether the slave can handle multiple requests at a time or not. I not the only one request is sent at a time and the answer will be awaited before sending a new request.

Modbus_Module

Changes as follows below:

UnitId

New. Usually has no meaning but some slaves need it to be set to something else than zero '0'.

Continous

New. Defines whether the request should be handled every cycle or only when trigged.

SendOp

New. Request to execute action defined by FunctionCode-attribute once. This attribute is only valid if Continous-attribute is set to 'No'.

Upgrade procedure

The upgrading has to be done from V4.5.0. If the project has a lower version, the upgrade has to be performed stepwise following the schema

V2.1 -> V2.7b -> V3.3 -> V3.4b -> V4.0.0 -> V4.1.3 ->V4.2.0->V4.5.0->V4.6.0

The upgrade procedure is to change the version of the project in the projectlist, and then activate Functions/Update Classes from the meny in the Configurator. If the V4.5 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. Fined 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. Change the version to V4.3.0. Save and close the administrator.

Update Classes

Do `sdf` to the project.

Start the Configurator, enter edit mode, and activate Functions/Update Classes in the menu.

Save and Build.